File No. S360-29
Order No. GC28-6594-7

OS

IBM Systems Reference Library

IBM System/360 Operating System

PL/I (F)

Programmer's Guide

Program Number 360S-NL-511

This publication is a companion volume to IBM System/360 Operating System: PL/I (F) Language Reference Manual, Form C28-8201. Together, the two books form a guide to the writing and execution of PL/I programs under the control of an IBM System/360 Operating System that includes the PL/I (F) Compiler. The Programmer's Guide is concerned with the relationship between a PL/I program and the operating system. It explains how to compile, link edit, and execute a PL/I program, and introduces job control language, the linkage editor, and other essential features of the operating system.

Eighth Edition (January, 1971)

This is a major revision of, and obsoletes, C28-6594-6 and Technical Newsletter GN33-6016. In addition to incorporating information from the Technical Newsletter this new edition contains changes to the description of the PL/I sorting facilities and a number of minor changes throughout. Changes to the text, and small changes to illustrations, are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol • to the left of the caption.

This edition applies to Release 20 of the IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 Bibliography SRL Newsletter, Form N20-0360, for the editions that are applicable and current.

The information contained in this publication concerning Model 195 support is for planning purposes only.

Requests for copies of IBM publications should be made to your IBM representative or the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM United Kingdom Laboratories Ltd., Programming Publications, Hursley Park, Winchester, Hampshire, England.

Preface

This publication and IBM System/360 Operating System: PL/I (F) Language Reference Manual form a complementary pair Programmer's Guide is concerned with the relationship between a PL/I program and IBM System/360 Operating System; it explains how to compile, link edit, and execute a PL/I program. The Programmer's Guide is concerned with the relationship between a PL/I program and IBM System/360 Operating System; it explains how to compile, link edit, and execute a PL/I program.

Part 1 is intended primarily for the casual (non-specialist) programmer or for the newcomer to IBM System/360; the reader is assumed to have only an elementary grasp of PL/I and an awareness of the basic concepts of electronic data processing[1]. 2 and 3 are designed for use either as reference material or for reading as an introduction to the features they describe.

PREREQUISITE PUBLICATION

The reader is assumed to have a working knowledge of PL/I; he should therefore be familiar with the material contained in the following publication:

IBM System/360 Operating System: PL/I (F) Language Reference Manual, Order No. GC28-8201

RECOMMENDED PUBLICATIONS

The text of the Programmer's Guide refers to the following publications for information outside its scope:

IBM System/360 Operating System:

PL/I (F) Compiler, Program Logic Manual, Order No. GY28-6800

PL/I Subroutine Library Program Logic Manual, Order No. GY28-6801

Advanced Checkpoint/Restart Planning Guide, Order No. GC28-6708

---

[1]For example, as described in Introduction to IBM Data Processing Systems, Order No. GC20-1684.

Concepts and Facilities, Order No. GC28-6535

Linkage Editor and Loader, Order No. GC28-6538

Job Control Language User's Guide, Order No. GC28-6703

Job Control Language Reference, Order No. GC28-6704

System Generation, Order No. GC28-6554

Utilities, Order No. GC28-6586

System Control Blocks, Order No. GC28-6628

Messages and Codes, Order No. GC28-6631

Job Control Language Charts, Order No. GC28-6632

Supervisor and Data Management Services, Order No. GC28-6646

Supervisor and Data Management Macro Instructions, Order No. GC28-6647

Programmer's Guide to Debugging, Order No. GC28-6670

Queued Telecommunications Access Method Message Processing Program Services, Order No. GC30-2003

Queued Telecommunications Access Method Message Control Program, Order No. GC30-2005

Sort/Merge, Order No. GC28-6543

In addition to the publications listed above, the following contain information that may be helpful to the user:

IBM System/360 Operating System:

Operator's Reference, Order No. GC28-6691

Operator's Procedures, Order No. GC28-6692

System Programmer's Guide, Order No. GC28-6550

Storage Estimates, Order No. GC28-6551

PL/I Subroutine Library, Computational Subroutines, Order No. GC28-6590

IBM System/360:

Principles of Operation, Order No. A22-6821

Contents

Figures

PART 1: Basic Programming with the PL/I (F) Compiler

Chapter 1: Introduction to the Operating System

In IBM System/360, programs are usually executed as part of a group of programs collectively termed an operating system. This chapter introduces IBM System/360 Operating System[1] (the operating system that includes the PL/I (F) compiler), and describes the job control language that enables programmers to define the requirements of their programs for the operating system. Chapter 2 illustrates the use of job control language for running a simple PL/I program. The two chapters are complementary; the first briefly describes the operating system and job control language, and the second demonstrates how to use them to execute a PL/I program. Chapter 3 introduces the concept of storage of data and shows how to use a simple data set.

IBM System/360 Operating System

IBM System/360 Operating System consists of a control program and a number of processing programs that together assist both the operator and the programmer in the use of IBM System/360. The operating system relieves the programmer of routine and time-consuming tasks by controlling the allocation of storage space and input/output devices. Through the language translators that may be included, it makes programming easier by permitting the use of high-level languages such as PL/I. And it increases the throughput of the machine because it can process a stream of jobs without interruption by the operator; it provides automatic transition from one job to another.

The control program supervises the execution of all processing programs and provides services that are required in common by the processing programs during their execution. It has four main elements:

1. Supervisor: The supervisor program is the control center of the operating system, and controls and coordinates all activity within it.

2. Master scheduler: The master scheduler forms a two-way communication link between the operator and the operating system.

3. Job scheduler: The job scheduler reads and analyzes the input job stream (the sequence of control statements and data entering the system), allocates input/output devices as necessary, initiates the execution of processing programs, and provides a record of the work processed.

4. Data management routines: The data management routines control input/output operations, regulate the use of input/output devices, and provide access to the data held in them.

The processing programs of the operating system include service programs (for example, the linkage editor) and language translators (for example, the PL/I (F) compiler) provided by IBM, as well as programs that are written by the user and incorporated as part of the system.

All the programs of the operating system are stored in system libraries, which are held in auxiliary storage on a direct-access storage device.

The most important components of the operating system that directly concern the PL/I programmer are the job scheduler, the PL/I (F) compiler, the linkage editor, and the linkage loader, all of which are discussed below. The operating system is described in IBM System/360 Operating System: Concepts and Facilities.

JOB SCHEDULER

The job scheduler is the component of the operating system that handles communications between the programmer and the services provided by the operating system. A simple programming language called job control language (JCL) enables the programmer to specify his requirements to the operating system. The statements of this language indicate to the job scheduler the start and name of the job, specify the programs that are to be executed, and define the auxiliary storage requirements of the programs. In response to the job control statements, the job scheduler allocates the input/output units required, notifying the operator of any tapes or disk

---

[1]IBM System/360 Operating System is frequently referred to as 'the operating system,' or simply 'the system.'

packs that must be mounted, and then requests the supervisor program to initiate the execution of the specified programs. After the execution of each program the job scheduler prints a record of the work done.

PL/I (F) COMPILER

The PL/I (F) compiler is a program that translates PL/I source programs into IBM System/360 machine instructions. The set of instructions produced by a compilation is termed an object module. An object module is not in a form suitable for loading into main storage and subsequent execution; first it must be processed by the linkage editor or the linkage loader. (Chapter 5 discusses the compiler and describes the object module it produces.)

LINKAGE EDITOR

The linkage editor is a program that converts object modules into a form suitable for loading into main storage for execution; a program in this form is termed a load module. The output (load module) from the linkage editor is always placed in a library, from which the job scheduler can load it for execution.

The linkage editor can combine separately produced object modules and previously processed load modules into a single load module. It can make changes to sections of a load module: only sections that are affected by the changes need be re-compiled. It also permits a program that is too large for the space available in main storage to be divided so that it can be loaded and executed segment by segment.

Chapter 6 discusses the linkage editor and the differences between object modules and load modules.

LINKAGE LOADER

The linkage loader is a program that converts object modules into load modules, loads them into main storage and executes them, all in one job step. It can combine object or load modules into a single load module for execution; this load module is always placed in main storage, never in a library.

The use of the linkage loader and the options available are discussed in Chapter 6.

Job Control Language Examples

The following discussion of the job control language is an overview. Job control language is fully described in IBM System/360 Operating System: Job Control Langauge User's Guide, and Job Control Language Reference; however the most significant parameters of the DD statement are also described in Appendix B.

Job control language is the means by which a programmer communicates with the job scheduler; it allows the programmer to describe the work he wants the operating system to do, and to specify the input/output facilities he requires. Only seven types of statement are involved, of which four are relevant to this discussion: the JOB statement, the execute (EXEC) statement, the data definition (DD) statement, and a delimiter statement.

The JOB statement identifies a job to the job scheduler. In IBM System/360 Operating System, a job is an independent request for the facilities of the operating system; it comprises one or more job steps. A job starts with a JOB statement and continues until the next JOB statement is encountered.

The EXEC statement identifies a job step to the job scheduler. A job step involves a request for the execution of a program. Job steps can be interrelated: data can be passed from one job step to the next, and the execution of one job step can depend on the successful execution of a preceding step. (No such relationship exists between jobs; they are independent of one another.) A job step starts with an EXEC statement and continues until the next EXEC or JOB statement is encountered.

DD (data definition) statements describe the input/output facilities required in a job step.

The delimiter (/*) separates data in the input stream from the succeeding job control statements.

FORMAT OF JOB CONTROL STATEMENTS

A job control statement consists of one or more 80-byte records. Since 80-column punched cards are the most common input

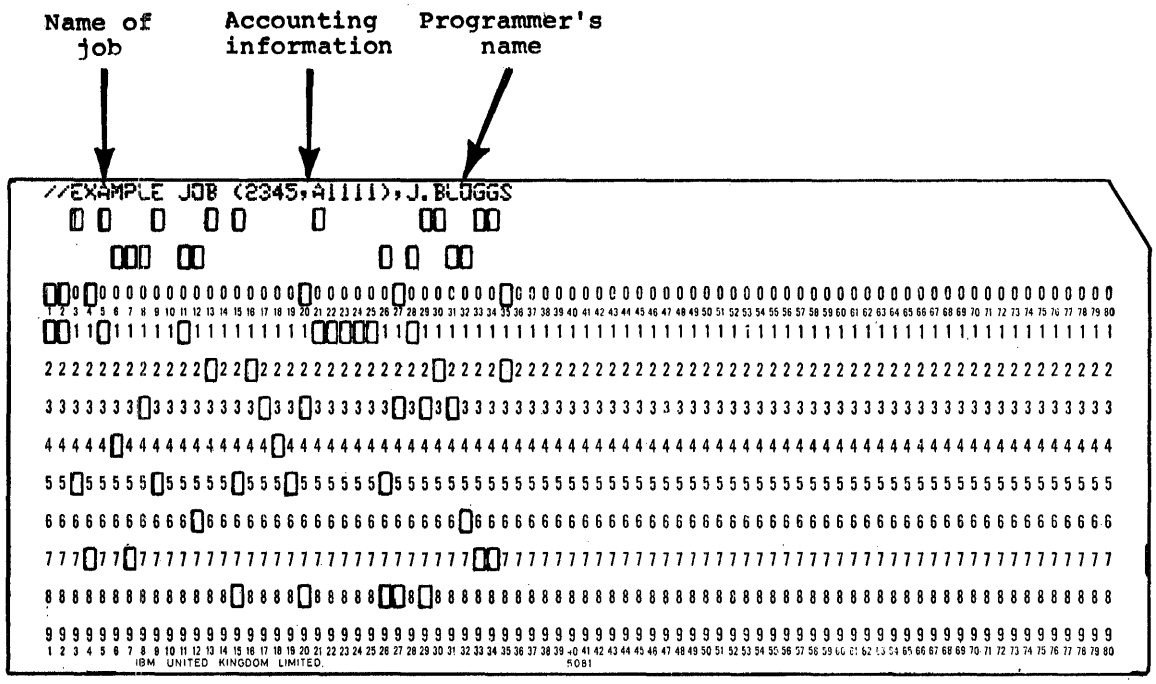


Figure 1-1. A JOB Statement

medium for the job stream, the following discussion refers to card columns rather than to bytes.

JOB, EXEC, and DD statements have the same format, examples of which appear in Figures 1-1, 1-2, and 1-3. These statements are identified by // in card columns 1 and 2. Each statement can contain four fields (name, operation, operand, comments), which are separated by one or more blanks; the name field starts in column 3. A job control statement must not extend beyond column 71; if necessary it can be continued on another card, as shown in Figure 1-3.

The name field, which begins in column 3, can range from one to eight characters in length, and can contain any alphameric (alphabetic or numeric) or national (@ $ #) characters. The first character must be alphabetic or national. This field is somtimes omitted. The name identifies the statement and enables other job control statements (or PL/I statements) to refer to it.

The operation field specifies the type of job control statement (JOB, EXEC, DD). Whether the name field is used or not, the operation field must be preceded by at least one blank.

The operand field can contain one or more parameters separated by commas; these parameters pass information to the job scheduler, and, for the JOB, EXEC and DD statements, are of two types, positional and keyword. Positional parameters must be placed at the beginning of the operand field, and are identified by their position relative to other parameters. If a positional parameter is omitted, its absence is indicated by a comma, unless it is the last positional parameter, when the comma is omitted. A keyword parameter consists of a keyword followed by an equals sign, which is followed by a single value or a list of subparameters; keyword parameters may appear in any order, and their omission need not be indicated.

The comments field is intended for programmer's notes. It has no fixed format, and can contain any information. The comments field is the only field that can include blanks.

The following paragraphs contain a general description of the functions of the JOB, EXEC, DD, and delimiter statements; these, and other job control statements, are discussed further under appropriate headings in later chapters. IBM System/360 Operating System: Job Control Language, User's Guide, and Job Control Language Reference, gives a full description of all the job control statements, their formats and parameters.

JOB STATEMENT

The JOB statement (Figure 1-1) indicates the start of a job and the end of the preceding job. It assigns a name to the new job, which is used by the job scheduler in the messages it passes to the operator and prints on the program listing. The parameters (none of which need appear unless your installation has made some of them mandatory) include the programmer's name and accounting information.

EXEC STATEMENT

The EXEC statement (Figure 1-2) marks the start of a new job step and the end of the preceding job step. It requests the job scheduler to fetch a load module from the system library (or from another library) and cause it to be executed; other load modules can be loaded dynamically during the execution of the first, but only one can be named in the EXEC statement. The first parameter (PGM=) names the program to be executed. Sometimes the EXEC statement passes information to the program that it calls (PARM parameter), and it may include accounting information and specify conditions for bypassing the job step.

DD STATEMENT

In IBM System/360, a collection of data held in an auxiliary storage device (a reel of magnetic tape, a disk pack, etc.,) is termed a <u>data set</u>. A DD statement (Figure 1-3) identifies a data set and describes its attributes. There must be a DD statement for each data set that is used or created in a job step. The DD statements are placed immediately after the EXEC statement for the step.

The parameters of the DD statement contain such information as the name of the data set, the name of the volume on which it resides, the type of device that holds the data set, the format of the records in the data set, and the method that will be used to create or access the data set. The name of the DD statement provides a symbolic link between the file named in a PL/I program and the actual name and location of the corresponding data set. The DD statement allows the programmer to leave the specification of his data set requirements until he is ready to execute his program, and enables him to relate the file in his program to different data sets on different occasions. Some data set information can be given in the PL/I ENVIRONMENT attribute rather that in a DD statement.

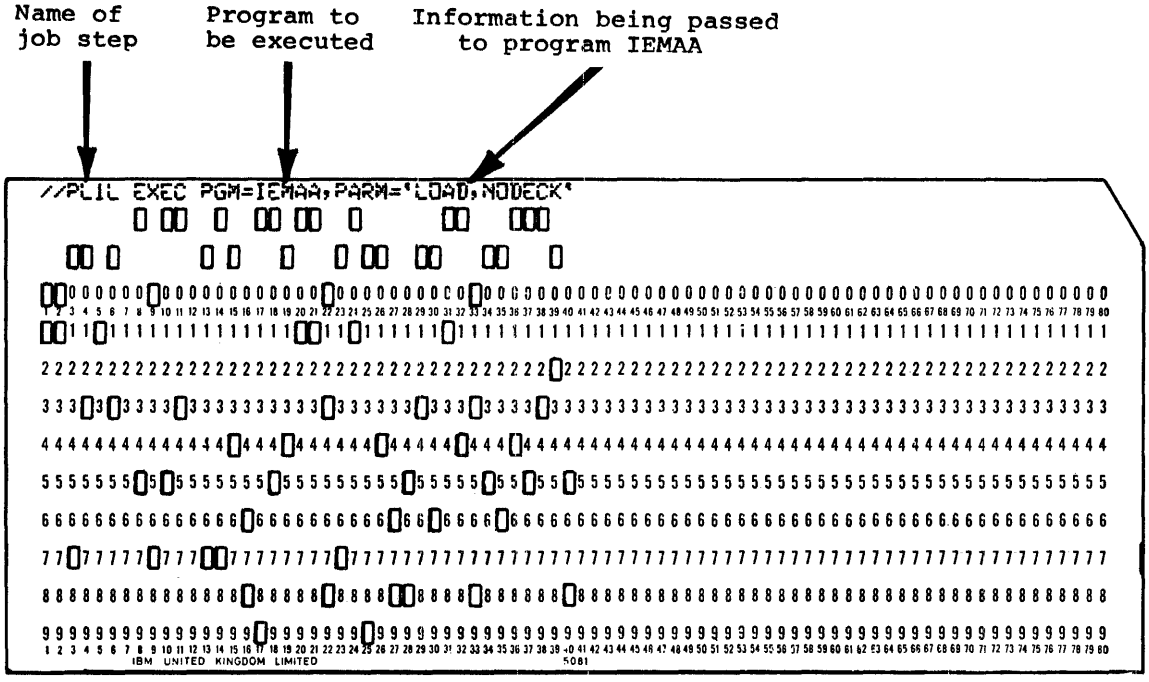


Figure 1-2. An EXEC Statement

Chapter 3 discusses the creation and access of simple data sets; Chapter 9 is a complete discussion of data management for a PL/I program.

DELIMITER STATEMENT

The delimiter statement consists of the characters /* in card columns 1 and 2. It separates data in the input stream from the job control statements that follow the data. Data in the input stream is usually preceded by a DD statement with the operand *, for example:

```
//SYSIN DD *
```

If the data includes cards that have // in the first two columns, it is preceded by a DD statement with the operand DATA, for example:

```
//SYSIN DD DATA
```

Executing a PL/I Program

A job consists of one or more job steps, and each job step normally uses one or more data sets. Thus the sequence of job control statements for a single job comprises a JOB statement followed by the EXEC and DD statements for each job step. Figure 1-4 illustrates such a sequence; it has been simplified by omitting all the parameters in the various statements.

The execution of a PL/I program requires two or three job steps:

1. Compilation: The compiler converts the PL/I statements into machine instructions, which form an object module.
2. Link-editing or link-loading: The linkage editor processes the object module produced in step 1, and forms it into a load module that can be executed. The linkage loader converts the object module into a load module and executes it.
3. Execution: The load module formed by the linkage editor is loaded into main storage and executed.

The sequence of job control statements in Figure 1-4 might be used for compiling, link-editing, and executing a PL/I program.

The JOB statement would probably have to include certain parameters required by the installation. The parameters needed and the values used with them vary from installation to installation; it is your responsibility to supply the correct information in this statement.

The EXEC statement for the first job step (named PL1L) requests the execution of the program IEMAA, the PL/I (F) compiler; the DD statements defining the data sets required for this step follow the EXEC statement. SYSPRINT and SYSIN refer to the printer and the card reader, respectively, as they do in the other job steps. The object module produced by the compiler is placed in the data set identified by SYSLIN. SYSUT1 and SYSUT3 define data sets used as workfiles.

The second EXEC statement (LKED) causes program IEWL, the linkage editor, to be executed. The linkage editor finds its primary input (the object module) in the data set referred to by the DD statement named SYSLIN; it may seek further input from the private library identified by SYSLIB. The load module produced by the linkage editor is placed in the data set identified by SYSLMOD. SYSUT1 defines a data set used as a workfile.

The last EXEC statement (GO) requests the execution of the load module created by the linkage editor in the previous step; a special form of the PGM parameter is used for this.

The sequence of job control statements using the linkage loader is shown in Figure 1-5.

Cataloged Procedures

Often the same set of job control statements is used over and over again (for example, to specify the compilation, link-editing, and execution of many different PL/I programs). To save programming time and to reduce the possibility of error, sets of standard series of EXEC and DD statements can be prepared once and 'cataloged' in a system library; such a set of statements is termed a cataloged procedure.

To retrieve a cataloged procedure, an EXEC statement is used in which the first parameter (PROC=) names the procedure. The effect is the same as if the job control statements of the cataloged procedure appeared in the job stream in the place of the EXEC statement that calls the procedure. In such an EXEC statement, the keyword PROC can be omitted; the name of the procedure stands alone as if it were a

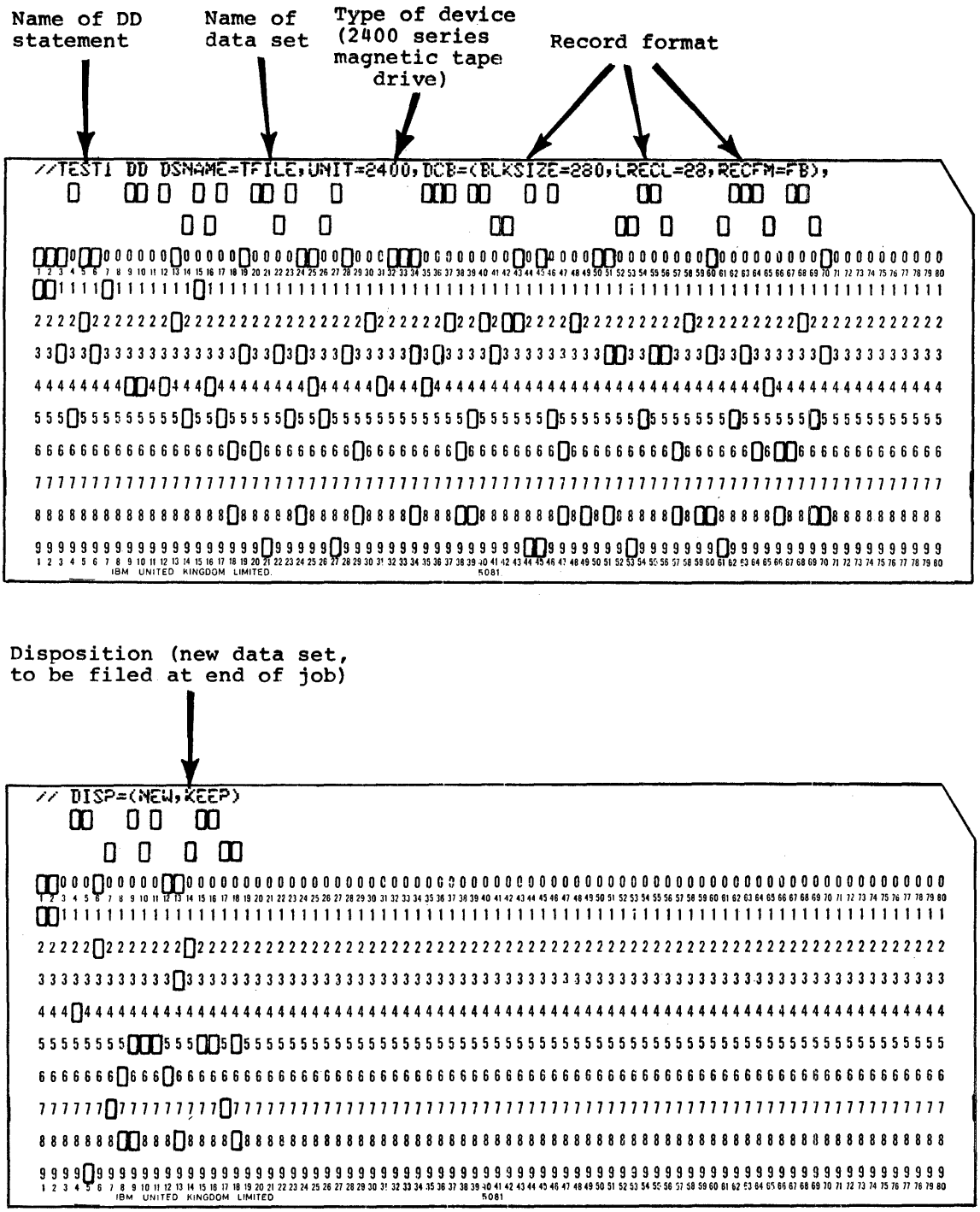


Figure 1-3. A DD Statement (Using a Continuation Card)

positional parameter. For example, the following EXEC statements have the same effect:

```
// EXEC PROC=PL1LFCLG

// EXEC PL1LFCLG
```

The job control statements in Figure 1-4 (apart from the JOB statement and the DD statements named SYSIN) are the same as those in the cataloged procedure PL1LFCLG. You can use this cataloged procedure, which is supplied by IBM, for compiling, link-editing, and executing a PL/I program. Chapter 2 illustrates the use of PL1LFCLG. Chapter 8 is a complete discussion of cataloged procedures; it describes the IBM cataloged procedures for PL/I and tells you how to modify them and how to catalog your own procedures.

```
//EXAMPLE JOB                    Start of job
//PL1L EXEC PGM=IEMAA             ┐
//SYSPRINT DD                     |
//SYSLIN DD                       |
//SYSUT1 DD                       |
//SYSUT3 DD                       |
//SYSIN DD *                       > First job
                                  |   step
Cards containing PL/I source      |
statements come here              |
                                  |
/*                                ┘
//LKED EXEC PGM=IEWL              ┐
//SYSLIB DD                       |
//SYSLMOD DD                       > Second job
//SYSUT1 DD                       |   step
//SYSPRINT DD                     |
//SYSLIN DD                       ┘
//GO EXEC PGM=*.LKED.SYSLMOD      ┐
//SYSPRINT DD                     |
//SYSIN DD *                      |
                                  |
Cards containing data to           > Third job
be processed by the PL/I          |   step
program come here                 |
                                  |
/*                                ┘
//NEXT JOB                       Start of next
    .                             job
    .
    .
```

Figure 1-4. Typical Sequence of Job Control Statements for Compile, Link-Edit, and Execute Steps

```
//EXAMPLE JOB                    Start of job
//PL1L EXEC PGM=IEMAA             ┐
//SYSPRINT DD                     |
//SYSLIN DD                       |
//SYSUT1 DD                       |
//SYSUT3 DD                       |
//SYSIN DD *                       > First job
                                  |   step
Cards containing PL/I source      |
statements come here              |
                                  |
/*                                ┘
//GO EXEC PGM=LOADER              ┐
//SYSLIB DD                       |
//SYSLIN DD                       |
//SYSLOUT DD                      |
//SYSPRINT DD                     |
//SYSIN DD *                      |
                                  |
                                   > Second job
Cards containing data to          |   step
be processed by the PL/I          |
program come here                 |
                                  |
/*                                ┘
//NEXTJOB JOB                    Start of next
    .                             job
    .
    .
```

Figure 1-5. Typical Sequence of Job Control Statements for Compile and Load-and-Execute Steps

If a procedure is programmer-written, it can be tested as an in-stream procedure before it is placed in the procedure library. An in-stream procedure is a series of job control statements enclosed between a PROC and a PEND statement appearing in the job stream. An in-stream procedure can be executed any number of times during a job and has the same content restrictions as a cataloged procedure. For further information about in-stream procedures, refer to the publication IBM System/360 Operating System: Job Control Language Reference.

Chapter 2: How to Run a Simple PL/I Progam

For a PL/I program that uses only punched-card input and printed output, the job control statements shown in Figure 2-1 are sufficient[1]. Appendix A includes an example of a simple PL/I program that uses these statements.

```
//EXAMPLE JOB
//  EXEC PL1LFCLG
//PL1L.SYSIN DD *

Insert here the cards
containing your PL/I
source statements.

/*
//GO.SYSIN DD *

Insert here the cards
containing the data to
be processed by your
program.

/*
```

If your program requires no external data, omit these statements.

Figure 2-1. Job Control Cards for the Execution of a Simple PL/I Program

Job Control Statements

```
//EXAMPLE JOB
```

EXAMPLE is the name of the job. Your job name must not have more than eight alphameric or national characters; the first character must be alphabetic or national. No parameters are given for this statement. If any are needed they will depend on your installation; the minimum requirement is probably an account number and your name. Before writing the JOB statement, ensure that you are familiar with the conventions established by your installation for the JOB statement and its parameters.

```
//  EXEC PL1LFCLG
```

PL1LFCLG is the name of a cataloged procedure supplied by IBM. When the job scheduler encounters the name of such a procedure in an EXEC statement, it substitutes for the EXEC statement a series of job control statements that have been written previously and cataloged in a system library. The cataloged procedure PL1LFCLG contains three job steps:

PL1L: The PL/I (F) compiler processes your source statements and translates them into a set of machine instructions (an object module).

LKED: The linkage editor creates a load module from the object module produced by the compiler. A load module is a series of machine instructions that are in a form suitable for loading into main storage and subsequent execution; only load modules can be loaded and executed.

GO: The load module created in step LKED is loaded into main storage and executed.

```
//PL1L.SYSIN DD *
```

This statement indicates that the data to be processed in step PL1L follows immediately in the card deck. SYSIN is the name that the compiler uses to refer to the device on which it expects to find this data. (In this instance, the device is the card reader, and the data is your PL/I program.)

```
/*
```

This signifies the end of the data.

```
//GO.SYSIN DD *
```

This statement indicates that the data to be processed by your program (in step GO) follows immediately in the card deck.

```
/*
```

This statement marks the end of the data to be processed by your program.

Note: You could have used the IBM cataloged procedure PL1LFCG in place of PL1LFCLG. This procedure consists of two job steps:

PL1L: The PL/I (F) compiler processes the source statements and produces an object module.

GO: The object module is converted to a load module, loaded into main storage and executed.

---

[1]Chapter 9 contains a complete discussion of data management.

FURTHER INFORMATION

Chapter 8 describes the cataloged procedure PL1LFCLG and other PL/I cataloged procedures supplied by IBM. Chapters 5, 6, and 7 deal with the job steps (compile, linkage, and execute) that are included in PL1LFCG and PL1LFCLG.

Chapter 3: How to Create and Access a Simple Data Set

A data set is any collection of data in auxiliary storage that can be created or accessed by a program. It can be punched onto cards or a reel of paper tape; or it can be recorded on magnetic tape or on a direct-access device such as a magnetic disk or drum. A printed listing can also be a data set, but it cannot be read by a program.

Data sets that are created or accessed by PL/I programs must have one of three types of organization: CONSECUTIVE, INDEXED, or REGIONAL or must be a teleprocessing data set. The items of data in INDEXED and REGIONAL data sets are arranged according to 'keys' that you supply when you create the data sets. CONSECUTIVE data sets do not use keys; when you create such a data set, data items are recorded consecutively in the order in which you present them. You can read the data items from a CONSECUTIVE data set only in the order in which they were presented or, in the case of a data set on magnetic tape, in the order in which they were presented or in the reverse order. Teleprocessing data sets are organized as consecutive groups of data items.

This chapter explains how to create and access simple CONSECUTIVE data sets stored on magnetic tape or on a direct-access device. It is intended to provide an introduction to the subject of data management, and to meet the needs of those programmers who do not require the full input/output facilities of PL/I and IBM System/360 Operating System. Chapters 9, 10, and 11 contain a full explanation of the relationship between the data management facilities provided by PL/I and those provided by the operating system.

Using a Data Set

To create or access a data set, you must not only include the appropriate input and output statements in your PL/I program, but you must also supply certain information to the operating system in a DD statement. A DD statement describes a data set and indicates how it will be handled; the information it supplies enables the job scheduler to allocate the necessary auxiliary storage devices, and permits the compiler to use the data management routines of the operating system to transmit data.

IBM System/360 Operating System: PL/I (F) Language Reference Manual describes the input and output statements that you will need to use in your PL/I program. Essentially, you must declare a file (explicitly or contextually) and open it (explicitly or implicitly) before you can begin to transmit data. A file is the means provided in PL/I for accessing a data set, and is related to a particular data set only while the file is open; when you close the file, the data set is no longer available to your program. This arrangement allows you to use the same file to access different data sets at different times, and to use different files to access the same data set.

Contextual declaration and implicit opening are performed, where required, in any of the input/output statements GET, PUT, READ, WRITE, LOCATE, and REWRITE.

You must provide a DD statement for each data set that you will use in each job step. If you use the same data set in more than one job step, each step which refers to that data set must have a DD statement for the data set.

If you are using a cataloged procedure, such as PL1LFCG or PL1LFCLG (described in Chapter 2), the DD statement for any data set processed by your program must appear in job step GO, in which your program will be executed. To signify its inclusion in this job step, you must prefix the name of the DD statement with the name of the job step. (For example, //GO.LIST DD... would indicate a DD statement named LIST in step GO.) The DD statement for the data set in the input stream (e.g., GO.SYSIN), if it is used in a PCP system, must be the last DD statement in your card deck.

How to Create a Simple Data Set

When you create a new data set, you should supply the following information to the operating system:

	Parameter of DD Statement
Type of output device that will write or punch your data set.	UNIT=
Serial number of the volume (tape reel, disk pack, etc.) that will contain your data set.	VOLUME=SER=
Name of your data set.	DSNAME=
Format of the records in your data set.	DCB=
Amount of auxiliary storage required for your data set (direct-access devices only).	SPACE=
Disposition of your data set at the end of the job step.	DISP=

Note: You can use the abbreviations VOL for VOLUME and DSN for DSNAME.

To give this information in the DD statement, use the parameters listed above. Appendix B contains a description of these parameters; the following paragraphs discuss their use in creating a CONSECUTIVE data set. Figure 3-1 summarizes this discussion.

TYPE OF OUTPUT DEVICE (UNIT=)

You must always indicate the type of output device (magnetic tape or disk drive, card punch, printer, etc.) that you want to use to create your data set. Usually the simplest way to do this is to use the UNIT parameter, although for a printer or a card punch it is often more convenient to use one of the special forms of DD statement discussed at the end of this chapter.

In the UNIT parameter, you can specify either the type number of the unit (for example, 2311 for a disk drive) or the name of a group of devices (for example, SYSDA for any direct-access device). Appendix B includes a list of the valid type numbers; the group names are established for a system during system generation.

VOLUME SERIAL NUMBER (VOLUME=SER=)

A unit of auxiliary storage such as a reel of magnetic tape or a magnetic disk pack is termed a volume; a volume can contain one or more data sets, and a data set can extend to more than one volume. Each volume is identified by a serial number that is recorded within it (and usually printed on the label attached to it). Although a deck of cards, a printed listing, and a reel of paper tape can be considered to be volumes, they do not have serial numbers.

Storage Device	Parameters of DD Statement		
	When required	What you must state	Parameters
All	Always	Output device	UNIT= or SYSOUT=
		Block size[1]	DCB=BLKSIZE=
Direct access only	Always	Auxiliary storage space required	SPACE=
Direct access and standard labeled magnetic tape	Data set to be used by another job step but only required by this job	Disposition	DISP=
	Data set to be kept after end of job	Disposition	DISP=
		Name of data set	DSN=
	Data set to be on particular volume	Volume serial number	VOL=SER=

[1]Alternatively, you can specify the block size in your PL/I program by using the ENVIRONMENT attribute.

Figure 3-1. Creating a CONSECUTIVE Data Set: Essential Parameters of DD Statement

You need specify a volume serial number only if you want to place the data set on a particular volume. If you omit the VOLUME parameter, the job scheduler will print in your program listing the serial number of the volume on which it placed the data set.

The VOLUME parameter has several subparameters. To specify a volume serial number, you need only the SER (serial number) subparameter (for example VOLUME=SER=12354).

NAME OF DATA SET (DSNAME=)

You must name a new data set if you want to keep it for use in future jobs. If the data set is temporary (required only for the job in which it is created), you can still name it, but you need not; if you omit the DSNAME parameter, the operating system will assume that the data set is temporary, and will give it a temporary name. (Any name you give to a temporary data set must be prefixed with the characters &&; for example, DSNAME=&&TEMP.)

FORMAT OF THE RECORDS (DCB=)

You can give record-format information either in your PL/I program (ENVIRONMENT attribute or LINESIZE option) or in a DD statement. This discussion refers only to the DD statement, and does not apply if you decide to give the information in your program; refer to IBM System/360 Operating System: PL/I (F) Language Reference Manual for a description of the ENVIRONMENT attribute and the LINESIZE option.

The records in a data set must have one of three formats: F (fixed length), V (variable length), U (undefined length). F-format and V-format records can be blocked or unblocked; V-format records can be spanned.

In most cases, you must specify a block size. If you do not give a record size, unblocked records of the same size as the block size are assumed. Note that, if you are using a PRINT file to produce printed output, you do not need to specify a block size in your DD statement or in your PL/I program; in the absence of other information, the compiler supplies a default line size of 120 characters. If you do not state the record format, U-format is assumed (except for data sets associated with PRINT files, for which V-format is the default).

To give record-format information in a DD statement, use the subparameters RECFM (record format), BLKSIZE (block size), and LRECL (logical record length) of the DCB parameter. The DCB parameter passes information to the operating system for inclusion in the data control block, which is a table maintained by the data management routines of the operating system for each data set in a job step; it contains a description of the data set and how it will be used. If your DCB parameter includes more than one subparameter, you must enclose the list in parentheses. For example:

```
DCB=(RECFM=FB,BLKSIZE=1000,LRECL=50)
```

AUXILIARY STORAGE REQUIRED (SPACE=)

When you create a data set on a direct-access device, you must always indicate the amount of space that the data set will occupy. Use the SPACE parameter to specify the size and number of the blocks that the data set will contain. If you may want to extend the data set in a later job or job step, ensure that your original space allocation is sufficient for future needs; you cannot make a further allocation later. If the SPACE parameter appears in a DD statement for a non-direct-access device, it is ignored.

DISPOSITION OF DATA SET (DISP=)

If you want to keep a data set for use in a later job step or job, you must use the DISP parameter to indicate how you want it to be handled. You can pass it to another job step, keep it for use in a later job, or enter its name in the system catalog. If you want to keep the data set, but do not want to include its name in the system catalog, the operating system will request the operator to demount the volume in which it resides and retain it for you. If you do not include the DISP parameter, the operating system will assume that the data set is temporary and will delete it at the end of the job step.

The DISP parameter can contain three positional subparameters. The first indicates whether the data set is new or already exists, the second specifies what is to be done with it at the end of the job step, and the third indicates how it should be treated if the job step is terminated abnormally by the operating system. If you omit either of the first two, you must indicate its absence by a comma.

For example,

```
DISP=(,CATLG,DELETE)
```

indicates that the data set is to be cataloged if the job step terminates normally, and deleted if it is terminated abnormally; the omission of the first subparameter indicates that the data set is assumed by default to be new.

How to Access an Existing Data Set

When you want to read or update an existing data set, your DD statement should include information similar to that given when the data set was created. However, for data sets on labeled magnetic tape or on direct-access devices, you can omit several parameters because the information they contain was recorded with the data set by the operating system when the data set was created. Figure 3-2 summarizes the essential information.

Except in the special case of data in the input stream (described below), you must always include the name of the data set (DSNAME) and its disposition (DISP).

TYPE OF INPUT DEVICE

You can omit the UNIT parameter if the data set is cataloged or if it was created with DISP=(,PASS) in a previous step of the same job. Otherwise, it must always appear.

VOLUME SERIAL NUMBER

You can omit the VOLUME parameter if the data set is cataloged or if it was created with DISP=(,PASS) in a previous step of the same job. Otherwise it must always appear.

NAME OF DATA SET

The DSNAME parameter can either refer back to the DD statement that defined the data set in a previous job step, or it can give the actual name of the data set. (You would have to use the former method to refer to an unnamed temporary data set.)

FORMAT OF THE RECORDS

You must always state a block size for punched cards or paper tape; otherwise, record-format information is not required. Block size can also be specified in your PL/I program, using the ENVIRONMENT attribute.

AUXILIARY STORAGE REQUIRED

You cannot add to, or otherwise modify, the space allocation made for a data set when it was created. Accordingly, the SPACE parameter is never required in a DD statement for an existing data set.

Parameters of DD Statement			
When required		What you must state	Parameters
Always		Name of data set	DSN=
		Disposition of data set	DISP=
If data set not cataloged	All devices	Input device	UNIT=
	Magnetic tape and direct access	Volume serial number	VOL=SER=
For punched cards or paper tape		Block size[1]	DCB=BLKSIZE=

[1]Alternatively, you can specify the block size in your PL/I program by using the ENVIRONMENT attribute.

Figure 3-2. Accessing a CONSECUTIVE Data Set: Essential Parameters of DD Statement

DISPOSITION OF DATA SET

You must always include the DISP parameter to indicate to the operating system that the data set already exists. Code DISP=SHR if you want to read the data set, DISP=OLD if you want to read and/or overwrite it, or DISP=MOD if you want to add records to the end of it.

You need not code the second term of the DISP parameter if you want the data set to resume the status it had before the job step; existing data sets will continue to exist, and newly created data sets will be deleted.

SPECIAL-PURPOSE PARAMETERS

Three parameters of the DD statement have special significance in that they permit you to use a very simple form of DD statement; they are SYSOUT, which is particularly useful for printed or punched-card output, and *, and DATA, which allow you to include data in the input stream.

SYSTEM OUTPUT (SYSOUT=)

A system output device is any unit (but usually a printer or a card punch) that is used in common by all jobs. The computer operator allocates all the system output devices to specific classes according to device type and function. The usual convention is for class A to refer to a printer and class B to a card punch; the IBM-supplied cataloged procedures assume that this convention is followed.

To route your output via a system output device, use the SYSOUT parameter in your DD statement. The only essential additional parameter is the block size (if not already specified in your PL/I program by using the ENVIRONMENT attribute). Thus, if you want to punch cards, you can use the DD statement

```
//GO.PUNCH DD SYSOUT=B,DCB=BLKSIZE=80
```

DATA IN THE INPUT STREAM

A convenient way to introduce data to your program is to include it in the input job stream with your control statements. Data in the input stream must, like job control statements, be in the form of 80-byte records (usually punched cards), and must be immediately preceded by a DD statement with the single parameter * in its operand field, for example:

```
//GO.SYSIN DD *
```

To indicate the end of the data, include a delimiter job control statement (/*). A DD statement that introduces data in the input stream must be the last DD statement in the job step.

If your data includes records that commence // in the first two columns use the parameter DATA, for example:

```
//GO.SYSIN DD DATA
```

Standard Files

PL/I includes two standard files, SYSIN for input and SYSPRINT for output. If your program includes a GET statement without the FILE option, the compiler inserts the file name SYSIN; if it includes a PUT statement without the FILE option, the compiler inserts the name SYSPRINT.

If you use one of the IBM-supplied cataloged procedures to execute your program, you will not need to include a DD statement for SYSPRINT; step GO of the cataloged procedures includes the statement:

```
//SYSPRINT DD SYSOUT=A
```

Note that no block size is specified in this DD statement; the block size for the data set associated with SYSPRINT is supplied by the (F) compiler. However, if your program uses SYSIN, either explicitly or implicitly, you must always include a corresponding DD statement.

Examples

The examples of simple applications for CONSECUTIVE data sets shown in Figures 3-3 and 3-4 should need no further explanation. The first program evaluates the familiar expression for the roots of a quadratic equation and records the results in a data set on magnetic disk and on punched cards. The second program reads the disk data set created in the first and prints the results.

```
//J001PGEX JOB
//  EXEC PL1LFCLG
//PL1L.SYSIN DD *
 CREATE: PROC OPTIONS(MAIN);
         DCL PUNCH FILE STREAM OUTPUT,
             DISK FILE RECORD OUTPUT SEQUENTIAL,
             1 RECORD, 2(A,B,C,X1,X2) FLOAT DEC(6) COMPLEX;
         ON ENDFILE(SYSIN) GO TO FINISH;
         OPEN FILE(PUNCH), FILE(DISK);
 NEXT:   GET FILE(SYSIN) LIST(A,B,C);
         X1=(-B+SQRT(B**2-4*A*C))/(2*A);
         X2=(-B-SQRT(B**2-4*A*C))/(2*A);
         PUT FILE(PUNCH) EDIT(RECORD) (C(E(16,9)));
         WRITE FILE(DISK) FROM(RECORD);
         GO TO NEXT;
 FINISH: CLOSE FILE(PUNCH), FILE(DISK);
         END CREATE;
/*
//GO.PUNCH DD SYSOUT=B,DCB=BLKSIZE=80
//GO.DISK DD  UNIT=2311,VOLUME=SER=D186,DSNAME=ROOTS,
//            DCB=(RECFM=FB,BLKSIZE=400,LRECL=40),
//            SPACE=(TRK,(1,1)),DISP=(NEW,KEEP)
//GO.SYSIN DD *
5 12 4
4 -10 4
5 16 2
4 -12 10
5 12 9
29 -20 4
/*
```

Figure 3-3. Creating a Simple CONSECUTIVE Data Set

```
//J027PGEX JOB
//COLEEX EXEC PL1LFCLG
//PL1L.SYSIN DD *
 ACCESS: PROC OPTIONS(MAIN);
         DCL RESULTS FILE RECORD INPUT SEQUENTIAL,
             1 RECORD, 2(A,B,C,X1,X2) FLOAT DEC(6) COMPLEX;
         ON ENDFILE(RESULTS) GO TO FINISH;
         PUT FILE(SYSPRINT) EDIT('A','B','C','X1','X2')
           (X(7),3(A,X(23)),A,X(22),A);
         OPEN FILE(RESULTS);
 NEXT:   READ FILE(RESULTS) INTO(RECORD);
         PUT FILE(SYSPRINT) SKIP EDIT(RECORD) (C(F(12,2)));
         GO TO NEXT;
 FINISH: CLOSE FILE(RESULTS);
         END ACCESS;
/*
//GO.RESULTS DD UNIT=2311,VOLUME=SER=D186,DSNAME=ROOTS,DISP=(OLD,KEEP)
```

Figure 3-4. Accessing a Simple CONSECUTIVE Data Set

PART 2: Using all the Facilities of the PL/I (F) Compiler

Chapter 4: Job Initialization

Introduction

The operating system requires certain preliminary information about a job in order to be able to process it. For example, it must be able to recognize the beginning and end of a job, and it requires details of the job environment (for example, which control program is used). Most of this information is provided in the job control language; the remainder is either information already known to the operator because it is established for your installation, or information you will have to give the operator for your particular job. For example, if you have a choice of control programs, you must tell the operator which one you want.

The information given in the job control language is provided in the JOB statement and its parameters. The JOB statement indicates the beginning of a job and (in batch processing) the end of a previous job; the parameters provide information about the job environment. Full details of the purpose and syntax of this statement are given in IBM System/360 Operating System: Job Control Language User's Guide, and Job Control Language Reference. The use of the JOB statement and its parameters is described briefly here, together with job scheduling and the types of control program available.

JOB Statement

The JOB statement is always the first statement in your job. It identifies the job to the operating system; in particular, it identifies the job to the job scheduler so that the latter can begin job processing. The job scheduler has three main components:

1. Reader/interpreter: This checks the job control language and (if required) stores the data from the input stream on a direct-access device and places control information about the job in an input queue.

2. Initiator/terminator: This selects the next job step for execution, and allocates devices and resources. After processing, it terminates the job step. After processing of the last job step, it terminates the job.

3. Output writer (MFT or MVT): This handles the transmission of data from output data sets to a system output device (such as a printer or a card-punch).

The JOB statement also provides information on:

1. Work control; for example, accounting information and the programmer's name.

2. Job environment; for example, information relating to the control program under which the job will be executed.

Note that the JOB statement does not select the control program for you; it merely allows you to specify information required by the control program you are using. The information to be given in the job statement fields is described below.

Name Field

A valid job name must appear in this field. Certain words must not be used as job names as they are command statements used by the operator to communicate with the operating system. Examples of these are SPACE and JOBNAMES; you must find out whether these or other names are used as command statements at your installation.

Jobs being executed concurrently should have different job names.

Operation Field

The word JOB must appear in this field.

Operand Field

The full set of positional and keyword parameters used with the JOB statement is:

Positional

Accounting information

Programmer's name

Keyword

CLASS

COND

MSGCLASS

MSGLEVEL

PRTY

REGION

ROLL

TYPRUN

The use of these is discussed in the IBM System/360 Operating System: Job Control Language Reference.

Although all these parameters are optional as far as the operating system is concerned, some or all of them will be mandatory at your installation, and some will not be available to you. Therefore, before using any of them, you need to know:

1. Which parameters are mandatory at your installation.
2. Which parameters are optional at your installation.
3. What happens if you omit an optional parameter that is required for a particular job. If there is no default for this parameter, the job could terminate at this point.
4. What happens if you include an optional parameter that is not required for a particular job. The parameter might be ignored or the job could terminate.

Note: The examples given in this manual omit the parameters of the JOB statement because of this installation dependence.

Job Scheduling

JOB SCHEDULING AND THE CONTROL PROGRAM

The operating system uses one of two forms of scheduling to process your job: sequential scheduling or priority scheduling. The type of scheduling employed for your job depends on the control program used.

Sequential Scheduling

Each job is processed in the order in which it exists in the input stream. Only one job at a time can be processed; all other jobs in the input stream must wait until this job is finished. Each job is executed as a single task.

Sequential scheduling is used by the primary control program (PCP).

Priority Scheduling

The jobs in the input stream are placed in input queues and selected for processing according to previously determined priorities. Jobs are processed in a multiprogramming environment, that is, several jobs can be processed concurrently. Each job step can be executed as a single task or as several tasks:

1. One job step, one task: Each job step is executed as a task. The number of tasks that can exist concurrently is fixed, and is restricted to the number selected for execution (that is, tasks cannot be created during execution of the job step).

   Jobs are processed in this way with the MFT (multiprogramming with a fixed number of tasks) control program.

2. One job step, several tasks: Each job step is executed as a task. Additional tasks (subtasks) can be created dynamically during execution of the job step. Therefore the number of tasks that can exist concurrently is variable, and depends on the number of subtasks created.

   Jobs are processed in this way with the MVT (multiprogramming with a variable number of tasks) control program.

Control Program

The three control programs currently available in the operating system are:

PCP

MFT

MVT

Your installation may have only one of these control programs, or it may have two or all of them. You must find out which control program or programs are available to you and design your source program accordingly. To assist you in the choice (if you have one), a brief description of each control program is given below, together with the set of JOB statement parameters applicable to that program. Further details of the control programs are contained in the following publications:

IBM System/360 Operating System:

Concepts and Facilities

Operator's Reference

Operator's Procedures

Storage Estimates

PRIMARY CONTROL PROGRAM (PCP)

JOB Statement Parameters

Accounting information

Programmer's name

COND

MSGLEVEL

Input

The job stream is read in from a card or tape device, and is loaded into main storage. Only one input reader is available at one time.

Job Selection

Only one job can be processed at a time. The whole of main storage and all resources are allocated to that job; no other job can gain control until this job has completed execution.

Task Execution

The job scheduler programs are executed as tasks; each job step of the job to be processed is executed as part of a job scheduler task. These job steps are executed sequentially.

Output

System output is put out through the SYSOUT stream; problem program output can be put out through the SYSOUT stream or a user data set. Up to eight output writers are available; each one writes one class of output onto one device (such as a printer). Each output class is designated by one of the letters A through Z. Class A is the standard system output class; there must always be a device available for this class.

MFT CONTROL PROGRAM

JOB Statement Parameters

Accounting information

Programmer's name

CLASS

COND

MSGCLASS

MSGLEVEL

PRTY

TYPRUN

Input

The job stream is read from a card or a tape device, or from a direct-access device, and is stored on a direct-access device. If there is any error in the job control statements for a job, that job is terminated. Up to three input readers are available.

Job Selection

Jobs are placed in an input queue; up to fifteen input queues are available. The queue selected depends on the job class, as specified in the CLASS parameter of the JOB statement. A job is placed in an input queue according to its priority, as

specified in the PRTY parameter of the JOB statement.

A job with the highest priority is placed at the head of the queue; that with the lowest at the end. A job with a priority the same as a job already in the queue, is placed immediately behind that job.

If a job is not given a priority, a default priority is given to it by the system, and it is placed in an input queue at the appropriate place.

In MFT, main storage is divided into partitions. One partition is allocated to one job at a time. Partitions are independent units of main storage. The work being done in one partition cannot affect work being done in another partition; data cannot be left in a partition for use by the next job step to be loaded.

Up to fifteen partitions are available for user jobs (there are other partitions but they are not available to you); therefore up to fifteen jobs can be processed concurrently. The number of partitions at a given time, and the size of each one, may be fixed for a particular installation, but usually both the number and size can be selected by the operator. The minimum size of a user partition is 8K bytes. You must find out the conventions on partitions at your installation.

Partitions are arranged in order of priority. The partition with the highest main-storage address has the highest priority; partitions with successively lower addresses have successively lower priorities. A task is the work executed in a particular partition. Partitions, not job steps, compete for control; if a job step enters a wait state or completes execution, control passes to the partition waiting with the highest priority. Neither job class nor job priority have any affect on competition between tasks for control.

The selection (by the initiator) of jobs for execution depends on the relationship between the partitions and the input queues (that is, the job classes). A partition can service up to three job classes; a job class can be allocated to more than one partition. If a partition services more than one job class, it searches those classes for jobs in a predetermined order. For example:

Partitions:		
	Highest priority	P0
		P1
		P2
	Lowest priority	P3

Class	Job	Class	Job	Class	Job
A	1	B	1	C	1
	2		2		2
	3		3		3

Allocations:

Class A:	P0, P1
Class B:	P0, P1, P2
Class C:	P2, P3
Partition P0:	A, B
Partition P1:	A, B
Partition P2:	B, C
Partition P3:	C

In this configuration, when the job step in P0 completes execution, class A is searched to see if there are any jobs waiting for selection. If there are not, class B is searched. Similarly P1 searches A and then B, P2 searches B and then C, and P3 searches C only. A job in class A can be selected for P0 or P1, whichever is the first available; a job in B can be selected for P0, P1 or P2, and a job in C for P2 or P3.

Task Execution

Once a job has been selected for processing, its job steps are executed sequentially. When a job step enters a wait state, control passes to the job step waiting in the next highest priority partition, irrespective of the job class of the waiting job step.

Other Considerations

Your installation may have the time-slicing facility. Execution time is divided ('sliced') into a number of discrete periods. One period is allocated to one job, the next period to another job, and so on, among all the jobs to be processed. The total execution time for one job is interleaved with that for other jobs, and thus all jobs are kept moving. Usually one particular class is reserved for time-sliced jobs; if your job is to be time-sliced you must find out the time-slicing class at your installation. Similarly, if you use the TYPRUN parameter, one class may be reserved for jobs whose processing is to be delayed.

If your installation has a shared PL/I library, this could affect the amount of space available in a partition or even the partition size.

Output

The output stream from the job is stored temporarily on a direct-access device. The output data is stored in an output queue; each member of the queue is associated with an output class, as designated in the SYSOUT parameter. If system and problem program data are in the same class, they are placed in the queue member in the order:

System messages at job initiation

Problem program data

System messages at job termination

Up to 36 output queues are available. The contents of each output class are written out by a system output writer onto the device (printer, punch, or tape) associated with that class. Up to 36 system output writers are available; each output writer can service up to eight output classes.

MVT CONTROL PROGRAM

JOB Statement Parameters

Accounting information

Programmer's name

CLASS

COND

MSGCLASS

MSGLEVEL

PRTY

REGION

ROLL

TYPRUN

Input

The job stream is read from a card or a tape device, or from a direct-access device, and is stored on a direct-access device. If there are any errors in the job control statements for a job, that job is terminated. As many input readers as are required can read input streams.

Job Selection

Jobs are placed in a single input queue, and are arranged within the queue according to their priorities, as specified in the PRTY parameter of the JOB statement.

A job with the highest priority is placed at the head of the queue; that with the lowest at the end. A job with a priority the same as a job already in the queue, is placed immediately behind that job.

If a job is not given a priority, a default priority is given to it by the system, and it is placed in the input queue at the appropriate place.

In MVT, main storage is divided into regions. The amount of storage required for a region is specified in the REGION parameter. The number of regions that can exist concurrently depends on the total size of main storage and on the region sizes specified. Each job is executed in one region. Regions are independent units of main storage; work being done in one region cannot affect work being done in any other region. Data cannot be left in a region for use by the next job step to be loaded.

A region is not a permanent division of main storage. It exists only for the duration of the job step for which it was created. When a job is selected for execution, a region is created for it. A region does not have a priority and does not compete for control with other regions; such competition is between the tasks in the regions.

The selection of jobs for execution from the input queue depends on the job class and priority. Jobs are queued in order of priority or, if they have equal or no priority, in order of arrival in the queue. Job class has no affect on job position in the input queue. Jobs are selected by one of several initiators. When an initiator is allocated to a job class, it selects

only the jobs in that class; the order of taking them is determined by the job priority. An initiator can be allocated to more than one job class, and will search the classes in predetermined order. Up to fifteen job classes are available; the number of initiators depends on the number of job classes allocated to each one.

For example, if there are three initiators:

Init(A) for jobs in class	A
Init(B)	B
Init(C)	C

and there are eleven jobs in the input queue as follows:

Job Class	Priority
B	10
B	9
A	8
A	8
B	7
C	4
A	3
C	2
C	1
A	0
A	0

then, when the job scheduler requires the next job for processing, if Init(A) is the first initiator to be given control, it ignores the two B jobs at the head of the queue and selects the first A8 job. The next time Init(A) receives control it selects the second A8 job (assuming the contents of the queue have remained the same) and so on through all the class A jobs: A3, A0, A0. When Init(A) cannot find any jobs to select, Init(B) is given control and selects the class B jobs in the order: B10, B9, B7. Similarly, when Init(C) receives control, it selects the class C jobs in the order: C4, C2, C1.

More than two initiators can run together and one initiator can interrogate more than one class depending on the installation standards.

Task Execution

Once a job has been selected for processing, its job steps are executed sequentially; each job step is a task.

Each task competes for control with all the other tasks being executed. Each task has a limit priority that depends on:

1. The value in the PRTY parameter in the JOB statement.

2. The value in the DPRTY parameter in the EXEC statement for that job step.

3. If neither of these is specified, a default value is supplied by the job scheduler.

Each task also has a despatching priority; this can be changed during execution but cannot be greater than the limit priority. The tasks compete for control on the basis of the current values of their despatching priorities. Note that in MVT a priority can be specified on the EXEC statement; this does not affect the sequential execution of the job steps in a job but can affect considerably the chances of an existing task gaining control.

Once a task is being executed, it can create subtasks dynamically.A subtask has a limit and despatching priority in the same way as the originating task, and competes for control with all the tasks and subtasks being executed.

Other Considerations

If your installation has the time-slicing facility described above in 'MFT Control Program,' note that usually one particular priority is reserved for time-sliced jobs. Similarly, priorities are reserved for jobs whose processing is delayed by use of the TYPRUN parameter; and for the message control or message processing programs in teleprocessing. You must find out the conventions at your installation. If your installation has a shared PL/I library, this could affect the region size. Region sizes can be affected during execution by rollout/rollin.

Output

MVT output follows the same conventions as those for MFT output. If your installation has a universal character set (UCS) printer that will be used as an output writer, you must assign a separate output class to each character set image in the system library.

Chapter 5: Compilation

Introduction

The PL/I (F) compiler translates PL/I source statements into machine instructions. A set of machine instructions such as is produced by the compiler is termed an object module. (If appropriate control statements are inserted among the PL/I source statements, the compiler can create two or more object modules in a single run by means of batch compilation.)

However, the compiler does not generate all the machine instructions required to represent the source program; instead, for frequently used standard routines such as those that handle the allocation of main storage space and the transmission of data between main and auxiliary storage, it inserts references to standard subroutines that are stored in the PL/I subroutine library. An object module produced by the compiler is not ready for execution until the appropriate library subroutines have been included; this is the task of an operating system service program, the linkage editor or the linkage loader, which is described in Chapter 6. A module that has been processed by the linkage editor or linkage loader is termed a load module.

While it is processing a PL/I source program, the compiler produces a listing that contains information about the source program and the object module derived from it, together with diagnostic messages relating to errors or other conditions detected during compilation. Much of this information is optional, and is supplied only in response to a request made by including appropriate 'options' in the PARM parameter of the EXEC statement that requests execution of the compiler.

The compiler also includes a facility, the preprocessor or compile-time processor, which can modify the source statements or insert additional source statements before compilation commences.

COMPILATION

The compiler comprises a control module that remains in main storage throughout compilation, and a series of subroutines (termed phases) that are loaded and executed in turn under the supervision of the control module. Each phase performs a single function or a set of functions, and is loaded only if the services it provides are required for a particular compilation. The control module selects the appropriate phases in accordance with the content of the source program and the optional compiler facilities that you select. Figure 5-1 is a simplified flow diagram of the compiler.

The data that is processed by the compiler is known throughout all stages of the translation process as text. Initially, the text comprises the PL/I source statements submitted by the programmer; at the end of compilation, it comprises the machine instructions that the compiler has substituted for the source statements, to which is added some reference information for use by the linkage editor.

The source program must be in the form of a data set identified by a DD statement with the name SYSIN; frequently, the data set is a deck of punched cards. The source text is passed to the read-in phase either directly or via one of two preprocessor phases:

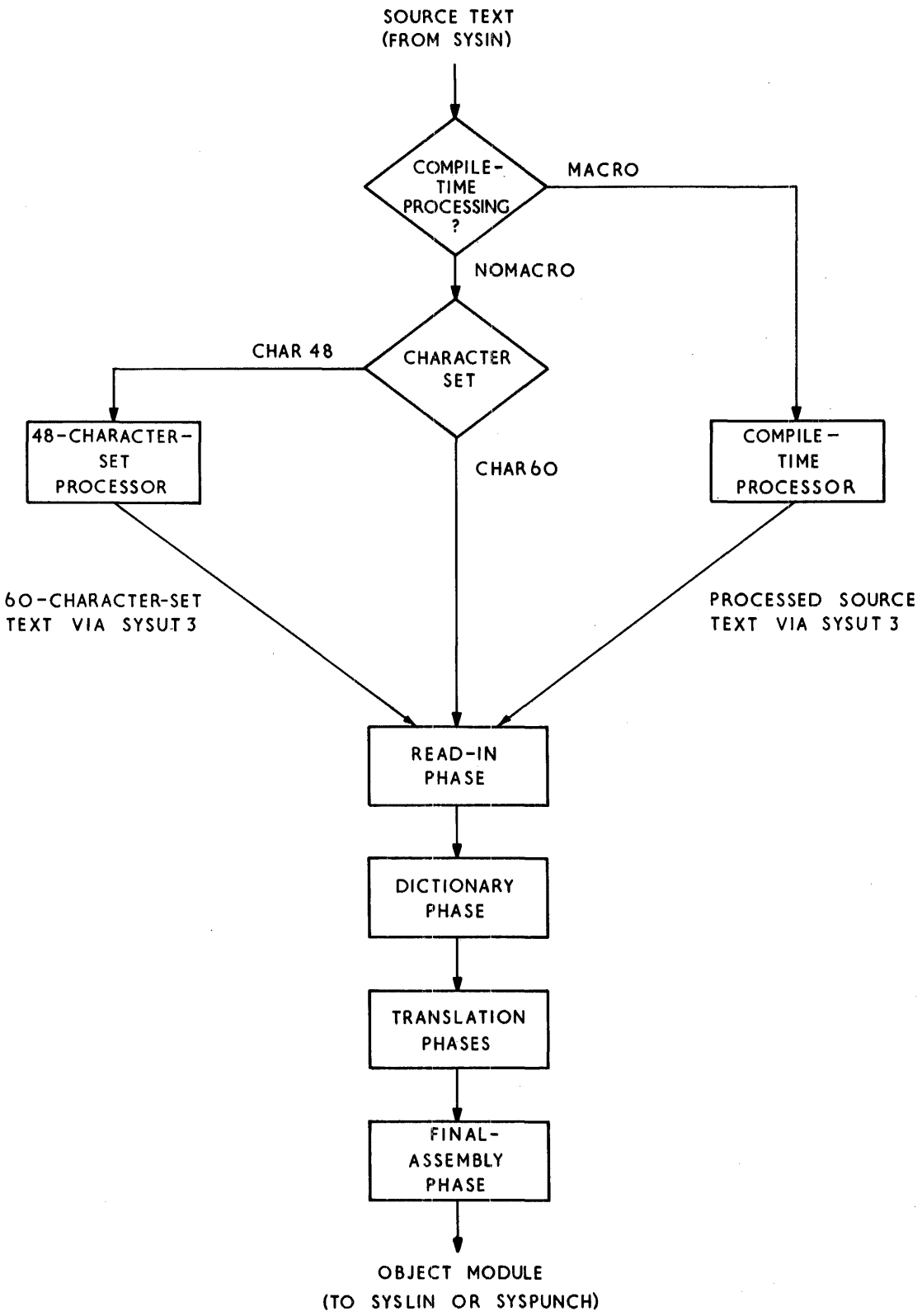
1. If the source text is in the PL/I 48-character set, the 48-character-set processor translates it into the 60-character set. You must indicate the need for translation by specifying the CHAR48 option.[1]

2. If the source text contains preprocessor statements, the compile-time-processor phase executes these statements in order to modify the source program or introduce additional statements. The compile-time processor includes a facility for translating statements written in the 48-character set into the 60-character set. To request the services of the compile-time processor, specify the MACRO option.

Both preprocessors place the translated source text in the data set defined by a DD statement with the name SYSUT3. The read-in phase takes its input either from this data set or from the data set defined by the DD statement SYSIN. This phase

---

[1]The compiler options are discussed under 'Optional Facilities,' later in this chapter.



igure 5-1. PL/I (F) Compiler: Simplified Flow Diagram

checks the syntax of the source statements and removes any comments and nonsignificant blank characters.

After read-in, the dictionary phase of the compiler creates a dictionary that contains entries for all the identifiers in the source text. The compiler uses the dictionary to communicate descriptions of the elements of the source program and the object program between phases. The dictionary phase of the compiler replaces all identifiers and attribute declarations in the source text with references to dictionary entries.

Translation of the source text into machine instructions involves several compiler phases. The sequence of events is:

1. Rearrangement of the source text to facilitate translation (for example, by replacing array or structure assignments with DO loops that contain element assignments).
2. Conversion of the text from the PL/I syntactic form to an internal syntactic form.
3. Mapping of arrays and structures to ensure correct boundary alignment.
4. Translation of text into a form similar to machine instructions; this text form is termed pseudo-code.
5. Storage allocation: the compiler makes provision for storage for STATIC variables and generates code to allow AUTOMATIC storage to be allocated during execution of the object program. (The PL/I library subroutines handle the allocation of storage during execution of the object program.)

The final-assembly phase translates the pseudo-code into machine instructions, and then creates the external symbol dictionary (ESD) and relocation dictionary (RLD) required by the linkage programs. The external symbol dictionary is a list that includes the names of all subroutines that are referred to in the object module but are not part of the module; these names, which are termed external references, include the names of all the PL/I library subroutines that will be required when the object program is executed. The relocation dictionary contains information that enables absolute storage addresses to be assigned to locations within the object module when it is loaded for execution. Chapter 6 contains a fuller discussion of the external symbol dictionary and the relocation dictionary, and explains how the linkage programs use them.

Job Control Language for Compilation

Although you will probably use cataloged procedures rather than supply all the job control statements required for a job step that invokes the compiler, it is necessary to be familiar with these statements so that you can make the best use of the compiler, and if necessary modify the statements of the cataloged procedures.

The IBM-supplied PL/I cataloged procedures for compilation are:

PL1DFC	Compile only (object module on punched cards).
PL1LFC	Compile only (object module stored on magnetic-tape or direct-access device).
PL1LFCL	Compile and link edit.
PL1LFCLG	Compile, link edit, and execute.
PL1LFCG	Compile, load, and execute.

Chapter 8 describes these cataloged procedures and how to modify or override the statements they contain.

The following paragraphs describe the essential job control statements for compilation; they use statements from the PL/I cataloged procedures as examples. Appendix B contains a description of the parameters of the DD statement that are referred to.

EXEC STATEMENT

The basic EXEC statement is:

```
// EXEC PGM=IEMAA
```

By using the PARM parameter of the EXEC statement you can select one or more of the optional facilities offered by the compiler; these facilities are described later in this chapter. The use of the other parameters of the EXEC statement is as described in Chapter 7, 'Executing the Load Module'.

Purpose	ddname	Associated Compiler Option
Primary input (PL/I source statements)	SYSIN	-
Punched card output	SYSPUNCH	DECK, MACDCK
Load module output	SYSLIN	LOAD
To contain overflow from main storage	SYSUT1	-
Storage for:	SYSUT3	
1. Converted source module when 48-character set used		CHAR48
2. Source statements generated by preprocessor		MACRO, COMP
Listing	SYSPRINT	-
Library containing source statements for insertion by preprocessor	SYSLIB	MACRO

Figure 5-2. Standard Data Sets for Compilation

DD STATEMENTS

The compiler requires several standard data sets, the number depending on the optional facilities that you request. You must define these data sets in DD statements with the standard names listed in Figure 5-2. The DD statements SYSIN and SYSPRINT are always required, and you should take the precaution of including SYSUT1 in case insufficient main storage is available to the compiler. In addition, if you specify any of the options listed in Figure 5-2, you must include the associated DD statement.

Figure 5-3 summarizes the characteristics of the compiler data sets. You can place any of them on a direct-access device; if it is likely that you will do so, include the SPACE parameter in the DD statements that define the data sets. The amount of storage space allocated in the standard cataloged procedures (Chapter 8) should suffice for most applications; however, Appendix H explains how to calculate the requirements for auxiliary storage.

Primary Input (SYSIN)

The primary input to the compiler must be a CONSECUTIVE data set containing PL/I source statements. These source statements must comprise one or more external procedures; if you want to compile more than one external procedure in a single run, you must separate the procedures in the input data set with *PROCESS statements (described under 'Batched Compilation' later in this chapter).

Eighty-column punched cards are commonly used as the input medium for PL/I source programs. However, the input data set may be on a direct-access device, magnetic tape, or paper tape. The data set may contain either fixed-length records, blocked or unblocked, or undefined-length records; the maximum record size is 100 bytes. The compiler always reserves 1000 bytes for two buffers for this data set; however, you may specify a block size of more than 500 bytes providing sufficient space is available to the compiler. (Use the SIZE option to allocate the additional space: refer to 'Optional Facilities,' later in this chapter.)

The standard PL/I cataloged procedures do not include a DD statement for the input data set; consequently, you must always provide one. The following example illustrates the statements you might use to compile, link-edit, and execute a PL/I program placed in the input stream:

```
//COLEGO JOB
// EXEC PL1LFCLG
//PL1L.SYSIN DD *

Insert here the source statements of
  your PL/I program

/*
```

Chapter 8 describes how to add DD statements to a cataloged procedure. Note

ddname	Possible Device Classes[1]	Record Format	Record Size (bytes)	Default Block Size (bytes)	Reserved Buffer Area (bytes)	No. of Buffers
SYSIN	SYSSQ or input job stream(specified by DD *)	F, FB, U	100 (max)	-	1000	2
SYSPUNCH	SYSSQ, SYSCP	F,FB	80	80	400	1
SYSLIN	SYSSQ	F,FB	80	80	400	1
SYSUT1	SYSDA	F	1024	-	-	
SYSUT3	SYSSQ	F, FB, U	80	-	160	
SYSPRINT	SYSSQ or SYSOUT device	V, VB	125	129	258	2
SYSLIB	SYSDA	F, FB, U	100 (max)	-	-	

[1]SYSSQ Magnetic-tape or direct-access device
SYSCP Card punch
SYSDA Direct-access device

Figure 5-3. Characteristics of Compiler Data Sets

that you must qualify the name of the added DD statement with the name of the job step within the cataloged procedure to which it refers (in this example, PL1L).

Output (SYSPUNCH,SYSLIN)

The compiler places the object module in the data set defined by the DD statement SYSLIN if you specify the option LOAD, and in the data set defined by SYSPUNCH if you include the option DECK; you may specify both options in one program. The object module is in the form of 80-byte fixed-length records, blocked or unblocked. The compiler always reserves 400 bytes for buffers for each of the output data sets; however, you may specify a block size of more than 400 bytes providing sufficient space is available to the compiler[1] (Use the SIZE option to allocate the additional space: refer to 'Optional Facilities', later in this chapter.)

The cataloged procedure PL1DFC includes the DD statement

```
//SYSPUNCH DD SYSOUT=B
```

This statement specifies that the data set is to be routed via the system output device of class B, which is usually a card punch. (However, the DD statement SYSPUNCH need not refer to a card punch.)

The other cataloged procedures that include a compilation job step contain the following DD statement:

```
//SYSLIN DD DSNAME=&&LOADSET,
//             DISP=(MOD,PASS),
//             UNIT=SYSSQ,
//             SPACE=(80,(250,100))
```

This statement defines a temporary data set named &&LOADSET on a magnetic-tape or direct-access volume; if you want to retain the object module after the end of your job, you must substitute a permanent name for &&LOADSET (i.e., a name that does not commence &&) and specify KEEP in the appropriate DISP parameter for the last step in which the data set is used. The term MOD in the DISP parameter allows the compiler to place more than one object module in the data set, and PASS ensures that the data set will be available to the next job step (link-edit) providing a corresponding DD statement is included there. The SPACE parameter allows an initial allocation of 250 eighty-byte records and, if necessary, 15 further allocations of 100 records (a total of 1750 records, which should suffice for most applications).

[1]The E-level linkage editor does not accept blocked records; specify blocked records for SYSLIN only if you are using the F-level linkage editor.

Workspace (SYSUT1, SYSUT3)

The compiler may require two data sets for use as temporary workspace. They are defined by DD statements with the names SYSUT1 and SYSUT3.

SYSUT1 defines a data set, known as the spill file, which the compiler uses for overflow text and dictionary blocks when compiling large source programs or when less than 57,344 bytes (56K bytes) of main storage are available for compilation. This data set must be on a direct-access device. It is good practice to include this DD statement even when you use the SIZE option to allocate more than 56K bytes to the compiler. The cataloged procedures include the following (or a similar) statement:

```
//SYSUT1 DD UNIT=SYSDA,
//          SPACE=(1024,(60,60),,CONTIG),
//          SEP=(SYSUT3,SYSLIN)
```

Although the SEP parameter is not essential, its employment increases the efficiency of access to the compiler data sets. You should never need to modify this DD statement.

The compiler requires the data set defined by SYSUT3 only when you use the 48-character set or when you employ the preprocessor. In each case, the compiler places the processed text on this data set before commencing compilation proper. All the cataloged procedures use the following DD statement:

```
//SYSUT3 DD UNIT=SYSSQ,
//             SPACE=(80,(250,250)),
//             SEP=SYSPRINT
```

Note that if a job being run under MVT has a number of job steps, and each job step requires a data set for use as temporary workspace, the result is a considerable overhead in time and space. To reduce this as far as possible, you can use dedicated workfiles. These are workspace data sets which are created by the initiator when the job is selected for execution. They can be used by each job step (in the job selected) that requires temporary workspace; they are deleted when the job is terminated.

To use dedicated workfiles in your job, you must first make sure that your installation has an initiator that can generate them. If it has, you can use these workfiles by specifying the ddname of the initiator workfile as the dsname of the workspace data set in your job stream; this data set must be specified as a temporary data set. For example, if an initiator has three dedicated workfiles as follows:

```
//SYSUT1  DD  (parameters)
//SYSUT2  DD  (parameters)
//SYSUT3  DD  (parameters)
```

then, if you want the workspace for the SYSUT1 and SYSUT3 data sets in a job step to be provided by the initiator workfiles, code:

```
//SYSUT1  DD  DSNAME=&&SYSUT1,...
//SYSUT3  DD  DSNAME=&&SYSUT3,...
```

The result is that this job step uses dedicated workfiles as workspace; the SYSUT1 and SYSUT3 DD statements in your job stream are ignored. The IBM-supplied cataloged procedures for PL/I include SYSUT1 and SYSUT3 DD statements with a dsname specified in this way.

There are several restrictions on the substitution of dedicated workfiles for workspace data sets in the job stream; for example, only direct-access devices are supported. You should consult your system programmer on the conventions and restraints that apply at your installation to each type of workspace data set.

Listing (SYSPRINT)

The compiler generates a listing that includes all the source statements that it processed, information relating to the object module, and, when necessary, diagnostic messages. Most of the information included in the listing is optional, and you can specify those parts that you require by including the appropriate compiler options. The information that may appear, and the associated options, are described under 'Listings', later in this chapter.

You must define the data set on which you wish the compiler to place its listing in a DD statement named SYSPRINT. The data set must have CONSECUTIVE organization. Although the listing is usually printed, it can be written on any magnetic-tape or direct-access device. For printed output, the following statement will suffice:

```
//SYSPRINT DD SYSOUT=A
```

The compiler always reserves 258 bytes for buffers for the data set defined by the DD statement SYSPRINT; however, you may specify a block size of more than 129 bytes provided sufficient main storage is available to the compiler. (Use the SIZE option to allocate the additional main

storage: refer to 'Optional Facilities', later in this chapter.)

Source Statement Library (SYSLIB)

If you use the preprocessor %INCLUDE statement to introduce source statements to your program from a library, you can either define the library in a DD statement with the name SYSLIB, or you can choose your own ddname (or ddnames) and specify a ddname in each %INCLUDE statement. (Refer to 'Compile-Time Processing', later in this chapter.) The DD statement SYSLIB is not included in the compilation job step of the standard cataloged procedures (and it has a different function in the link-edit step).

Note that for SYSLIB, the maximum record size permitted is 100 bytes and the maximum block size is 500 bytes.

EXAMPLE

The following example is a typical sequence of job control statements for compiling a PL/I program. The compiler options DECK and NOLOAD, which are described below, have been specified in order to obtain an object module as a card deck only. Chapter 6 includes a sequence of job control statements for link-editing such a card deck.

```
//COMP JOB
// EXEC PGM=IEMAA,PARM='DECK,NOLOAD'
//SYSPUNCH DD SYSOUT=B
//SYSUT1 DD UNIT=SYSDA,
//          SPACE=(1024,(60,60),,CONTIG)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *

Insert here the source statements to be
  compiled

/*
```

Optional Facilities

The (F) compiler offers a number of optional facilities that you can select by including the appropriate keywords in the PARM parameter of the EXEC statement that invokes it. The PARM parameter is a keyword parameter: code PARM= followed by the list of options, separating the options with commas and enclosing the list within single quotation marks[1]; for example:

```
// EXEC PGM=IEMAA,PARM='SIZE=72K,LIST'
```

The length of the option list must not exceed 100 characters, including the separating commas; however, many of the option keywords have an abbreviated form that you can use to save space. You may specify the options in any order.

If you are using a cataloged procedure, you must include the PARM parameter in the EXEC statement that invokes the procedure and qualify the keyword with the name of the procedure step that invokes the compiler; for example:

```
// EXEC PL1LFCLG,PARM.PL1L=('SIZE=100K',
//              L,E,A)
```

The compiler options are of two types:

1. Simple pairs of keywords: a positive form (e.g., LOAD) that requests a facility, and an alternative negative form (e.g., NOLOAD) that rejects that facility.

2. Keywords that permit you to assign a value to a function (e.g., SIZE=56K).

During system generation, your installation can specify for each option except OBJNM a default value that will apply if the option is not otherwise specified. For those options for which your installation does not allocate a default value, standard default values apply. Figure 5-4 lists all the compiler options with their abbreviated forms and the standard default values. The following paragraphs describe the options in five groups:

1. Control options, which establish the conditions for compilation (e.g., amount of main storage available, degree of optimization).

2. Preprocessor options, which request the services of the preprocessor and specify how its output is to be handled.

3. Input options, which specify the format of the input to the compiler.

4. Output options, which specify the type of data set that will contain the object module.

---

[1]If only one option appears, the quotation marks can be omitted.

Compiler Options		Abbreviated Names	Standard Defaults
Control options	SIZE=yyyyyy|yyyK|999999	SIZE	999999
	OPT=n	O	1
	STMT|NOSTMT	ST|NST	NOSTMT
	OBJNM=aaaaaaaa	N	-
	OBJIN|OBJOUT	OBJIN|OBJOUT	OBJOUT
	EXTDIC|NOEXTDIC	ED|NED	NOEXTDIC
	SYNCHKT|SYNCHKS|SYNCHKE	SKT|SKS|SKE	SYNCHKT
Preprocessor options	MACRO|NOMACRO	M|NM	NOMACRO
	COMP|NOCOMP	C|NC	COMP
	MACDCK|NOMACDCK	MD|NMD	NOMACDCK
Input options	CHAR60|CHAR48	C60|C48	CHAR60
	BCD|EBCDIC	B|EB	EBCDIC
	SORMGIN=(mmm,nnn[,ccc])	SM	(2,72)
Output options	LOAD|NOLOAD	LD|NLD	LOAD
	DECK|NODECK	D|ND	NODECK
Listing options	LINECNT=xxx	LC	50
	OPLIST|NOOPLIST	OL|NOL	OPLIST
	SOURCE2|NOSOURCE2	S2|NS2	SOURCE2
	SOURCE|NOSOURCE	S|NS	SOURCE
	NEST|NONEST	NT|NNT	NONEST
	ATR|NOATR	A|NA	NOATR
	XREF|NOXREF	X|NX	NOXREF
	EXTREF|NOEXTREF	E|NE	NOEXTREF
	LIST|NOLIST	L|NL	NOLIST
	FLAGW|FLAGE|FLAGS	FW|FE|FS	FLAGW
	DUMP[(argument-list)]	DP[(arg-list)]	-

• Figure 5-4. Compiler Options, Abbreviations, and Standard Defaults

5. Listing options, which specify the information to be included in the compiler listing.

CONTROL OPTIONS

SIZE

The SIZE option specifies the amount of main storage available for the compilation. Code this option in one of the following ways:

SIZE=yyyyyy specifies that yyyyyy bytes of main storage are available for the compilation. You need not supply leading zeros.

SIZE=yyyK specifies that yyyK bytes of main storage are available for the compilation (1K=1024 bytes). You need not supply leading zeros.

SIZE =999999 instructs the compiler to obtain as much main storage as it can. If it finds less than 45,056 bytes (44K bytes), it prints a message and attempts to continue compilation; this attempt may not, however, be successful. If you know that less than 48K bytes of main storage are available, do not specify SIZE=999999, but give the precise amount.

The following notes may help you select the optimum size for a compilation:

1. For compilation in an MFT partition or an MVT region, specify at least 8K bytes less than the partition or region size.

2. If you specify less than 57,344 bytes (56K bytes), the spill file (defined by the DD statement SYSUT1) will be opened.

3. Compilation speed is improved if the SIZE value is increased to the point where the spill file will not be opened. The SIZE value also determines the size of the dictionary and text blocks:

Main Storage Available (bytes)	Block Size (bytes)
45,056 - 57,343 (56K)	1,024
57,344 - 73,727 (72K)	2,048
73,728 - 135,167 (132K)	4,096
135,168 - 172,031 (168K)	8,192
172,032 or more	16,384

The available storage is that specified in the SIZE option less any space required for data set buffers (see note 4, below).

If the spill file has to be used either for text or dictionary blocks, diagnostic message IEM3898I will be printed. Note that spilled text blocks can be processed more efficiently than spilled dictionary blocks. The spill file will therefore be used for text blocks before it is used for dictionary blocks.

For each compilation, the message "AUXILIARY STORAGE WILL NOT BE USED FOR DICTIONARY WHEN SIZE=nnnk" is printed. Use of this size for a recompilation will provide a more efficient use of main storage but might result in the spilling of text blocks (and the production of message IEM3898I)

4. The compiler reserves part of the main storage available to it for use as data set buffers (intermediate storage areas for data transmitted between main and auxiliary storage). The compiler uses one buffer each for the data sets defined by SYSLIN and SYSPUNCH, and two buffers each for SYSPRINT and SYSIN; in each case, the size of the buffer is equal to the block size of the corresponding data set. If you specify a block size for any of the data sets that requires more buffer space than the compiler normally reserves, you should allow for the extra space in your SIZE option by adding the following quantities to the 44K bytes minimum required by the compiler:

   (2 * SYSIN block size) - 1000 bytes

   (2 * SYSPRINT block size) - 258 bytes

   SYSPUNCH block size - 400 bytes

   SYSLIN block size - 400 bytes

5. The text block size determines the total number of pairs of parentheses used for factoring attributes in a DECLARE statement:

Text Block Size (bytes)	Pairs of Factor Parentheses
1,024(1K)	146
2,048(2K)	292
4,096(4K)	585
8,192(8K)	1171
16,384(16K)	2340

OPT

The OPT option specifies the type of optimization required:

OPT=0 instructs the compiler to keep object-program storage requirements to a minimum at the expense of object-program execution time.

OPT=1 causes object-program execution time to be reduced at the expense of storage.

OPT=2 has the same effect as OPT=1, and in addition requests the compiler to optimize the machine instructions generated for certain types of DO-loops and expressions in subscript lists. IBM System/360 Operating System: PL/I (F) Language Reference Manual includes a discussion of DO-loop and subscript-expression optimization.

There is little difference in compilation time for optimization levels 0 and 1, but specifying OPT=2 could result in a substantial increase in compile time.

STMT or NOSTMT

The STMT option requests the compiler to produce additional instructions that will allow statement numbers from the source program to be included in diagnostic messages produced during execution of the compiled program.

The use of this option causes degradation of execution time. However you can get information about statement numbers and their associated offsets by referring to the TABLE OF OFFSETS. (See 'Options Used for the Compilation,' below.)

OBJNM

The OBJNM option allows you to name the load module that will be created by the linkage editor from the compiled object module. (If you do not specify a name, the linkage editor will use the member name from the DSNAME parameter of the DD statement SYSLMOD in the link-edit job step; see Chapter 6.) The option causes the compiler to place a linkage editor NAME statement at the end of the object module. The NAME statement has the effect of assigning the specified name to the preceding module when the module is link-edited. The format of the option is

OBJNM=aaaaaaaa

where 'aaaaaaaa' represents a name comprising not more than eight characters, the first of which must be an alphabetic character. The format of the resultant NAME statement, which is described fully in Chapter 6, is

bNAMEbaaaaaaaa(R)

where b represents a blank character.

The principal purpose of the OBJNM option is to facilitate the use of the linkage editor to create a series of load modules from the output of a batched compilation. (Refer to 'Batched Compilation', later in this chapter.) You can also use it to cause the linkage editor to substitute the new load module for an existing module of the same name in a library.

OBJIN or OBJOUT

You must specify the option OBJIN if you intend to execute the compiled program on an IBM System/360 Model 91 or 195. The special considerations for PL/I programs executed on Models 91 and 195 are discussed in Appendix G.

EXTDIC or NOEXTDIC

The EXTDIC option causes the compiler to use a dictionary with a capacity of 1.5 times that of the normal dictionary if the dictionary block size is 1K bytes, and 3.5 times that of the normal dictionary if the block size is greater than 1K bytes. This permits successful compilation of large programs that would otherwise overflow the dictionary capacity. As the use of EXTDIC reduces compilation speed, specify this option only when the source module cannot be compiled with the standard dictionary.

Programs that are large enough to require the EXTDIC option will be compiled very much more quickly if a large storage area is available. Ideally, enough storage should be available to hold the dictionary throughout compilation. As a rough guideline, the SIZE option should specify about 100,000 bytes plus 75 bytes for each identifier in the source module. Do not use the EXTDIC option when SIZE specifies less than 47,104 bytes.

SYNCHKT, SYNCHKS or SYNCHKE

This option specifies the conditions for termination after syntax checking if errors are detected. The option has three values specifying termination according to the severity of errors.

SYNCHKE terminates compilation if errors of severity ERROR or above are found during the syntax checking stages of compilation.

SYNCHKS terminates compilation if errors of severity SEVERE or above are found during the syntax checking stages of compilation.

With the SYNCHKS or SYNCHKE in effect, a message is written to SYSPRINT stating:-

'SYNTAX CHECK COMPLETED. COMPILATION CONTINUES' or 'SYNTAX CHECK COMPLETED. COMPILATION TERMINATED'

whichever is appropriate.

When using the value SYNCHKT compilation is terminated immediately an error of severity TERMINATION is encountered during the syntax check. In this case the syntax check is not completed, and therefore no special message is printed.

With SYNCHKT in effect the option is effectively turned off, and no special messages will be generated.

PREPROCESSOR OPTIONS

MACRO or NOMACRO

Specify MACRO when you want to employ the compiler preprocessor. The use of the preprocessor is described under 'Compile-Time Processing,' later in this chapter.

COMP or NOCOMP

Specify this option if you want the PL/I source module produced by the preprocessor to be compiled immediately. The source module is then read by the compiler from the data set identified by the DD statement SYSUT3.

MACDCK or NOMACDCK

Specify the option MACDCK if you want the output from the preprocessor in the form of a card deck. This output is written (punched) in the data set specified by the DD statement SYSPUNCH.

INPUT OPTIONS

CHAR60 or CHAR48

If the PL/I source statements are written in the PL/I 60-character set, specify CHAR60; if they are written in the 48-character set, specify CHAR48. IBM System/360 Operating System: PL/I (F) Language Reference Manual lists both character sets. (Note that the compiler will accept source programs written in either character set if you specify CHAR48.)

BCD or EBCDIC

The compiler will accept source statements in which the characters are represented by either of two codes; binary coded decimal (BCD) and extended binary-coded-decimal interchange code (EBCDIC). For binary coded decimal, specify the option BCD; for extended binary coded decimal interchange code, specify the option EBCDIC. Whenever possible, use EBCDIC since BCD requires translation and is therefore less efficient. IBM System/360 Operating System: PL/I (F) Language Reference Manual lists the EBCDIC representation of both the 48-character set and the 60-character set. The 029 Keypunch punches characters in EBCDIC form without multipunching; to obtain EBCDIC using the 026 you must multipunch some characters.

SORMGIN

The SORMGIN (source margin) option specifies the extent of the part of each input record that contains the PL/I source statements. The compiler will not process data that is outside these limits. The option can also specify the position of an ANS carriage control character to format the listing of source statements produced by the compiler if you include the SOURCE option. The format of the SORMGIN option is:

SORMGIN=(mmm,nnn[,ccc])

where mmm represents the number of the first byte of the field that contains the source statements,

nnn represents the number of the last byte of the source statement field, and

ccc represents the number of the byte that will contain the control character.

The value mmm must be less than or equal to nnn, and neither must exceed 100. The value ccc must be outside the limits set by mmm and nnn. The valid control characters are:

b	Skip one line before printing
0	Skip two lines before printing
-	Skip three lines before printing
+	Suppress space before printing
1	Start new page

Chapter 11 contains a full description of the use of printer control characters. If you do not specify a position for a control character, a default position defined by your installation may apply. You can nullify this default position by specifying the carriage control character to be zero (for example, SORMGIN=(1,72,0)).

If the value ccc is greater than the value set by the LRECL subparameter of the DCB parameter, the compiler may not be able to recognize it; consequently the listing may not have the required format. If the character specified is not a valid control character, a blank is assumed by default.

Source statements generated by the preprocessor always have a source margin (2,72). Columns 73-80 contain information inserted by the preprocessor; this information is described under 'Listing,' below.

OUTPUT OPTIONS

LOAD or NOLOAD

The LOAD option specifies that the compiler is to place the object module in the data set defined by the DD statement with the name SYSLIN.

DECK or NODECK

The DECK option specifies that the compiler is to place the object module, in the form of 80-column card images, in the data set defined by the DD statement with the name SYSPUNCH. Columns 73-76 of each card contain a code to identify the object module; this code comprises the first four characters of the first label in the external procedure represented by the module. Columns 77-80 contain a 4-digit decimal serial number: the first card is numbered 0001, the second 0002, etc.

LISTING OPTIONS

The listings produced by the compiler when you specify the following options are described under 'Listing' below.

LINECNT

The LINECNT option specifies the number of lines to be included in each page of a printed listing, including heading lines and blank lines. Its format is:

```
LINECNT=xxx
```

OPLIST or NOOPLIST

The OPLIST option requests a list showing the status of all the compiler options at the start of compilation.

SOURCE2 or NOSOURCE2

The SOURCE2 option requests a listing of the PL/I source statements input to the preprocessor.

SOURCE or NOSOURCE

The SOURCE option requests a listing of the PL/I source statements processed by the compiler. The source statements listed are either those of the original source program or the output from the preprocessor.

NEST or NONEST

The NEST option specifies that the source program listing should indicate for each statement, the block level and the level of nesting of a DO-group.

ATR or NOATR

The ATR option requests the inclusion in the listing of a table of source program identifiers and their attributes. Attributes with a precision of fixed binary (15,0) or less are flagged '*********'. An Aggregate Length Table, giving the length in bytes of all major structures and non-structured arrays in the source program, will also be produced when the ATR option is specified.

XREF or NOXREF

The XREF option requests the inclusion in the listing of a cross-reference table that lists all the identifiers in the source program with the numbers of the source statements in which they appear. If you specify both ATR and XREF, the two tables are combined. An Aggregate Lengh Table will also be produced when the XREF option is specified.

EXTREF or NOEXTREF

The EXTREF option requests the inclusion of a listing of the external symbol dictionary (ESD).

LIST or NOLIST

The LIST option requests a listing of the machine instructions generated by the compiler (in a form similar to System/360 assembler language instructions).

FLAGW or FLAGE or FLAGS

The diagnostic messages produced by the PL/I (F) compiler are graded in order of severity. The FLAG option specifies the minimum level of severity that requires a message to be printed:

FLAGW List all diagnostic messages

FLAGE List all diagnostic messages except 'warning' messages

FLAGS List only 'severe' errors and 'termination' errors

The severity levels are discussed under 'Listing,' below.

Dump

The DUMP option requests a formatted listing on SYSPRINT of the compiler modules, compiler storage, and compiler control blocks if an unrecoverable error is encountered. The DUMP option can also be used with optional arguments; the nature and purpose of these arguments are discussed in the publication IBM System/360 Operating System; PL/I (F) Compiler, Program Logic, Order No. GY28-6800.

This facility should only be used in the event of a compiler failure.

Listing

During compilation, the compiler generates a listing that contains information about the compilation and about the source and load modules. It places this listing in the data set defined by the DD statement SYSPRINT (usually output to a printer). The following description of the listing refers to its appearance on a printed page.

The listing comprises a small amount of standard information that always appears, together with those items of optional information requested in the PARM parameter of the DD statement that invoked the compiler or that were applied by default. Figure 5-5 lists the optional components of the listing and the corresponding compiler options.

The first page of the compiler listing is identified by the compiler version number and the operating system release number in the top left-hand corner, and by the heading OS/360 PL/I COMPILER (F) in the center. Starting with this page, all the pages of the listing are numbered sequentially in the top right-hand corner. On Page 1, immediately under the page number, the date of compilation is recorded in the form yy.ddd (yy=year, ddd=day). Page 1 also includes a statement of the options specified for the compilation, exactly as they are written in the PARM parameter of the EXEC statement.

The listing always ends with a statement that no errors or warning conditions were detected during the compilation or with one or more diagnostic messages. The format of the messages is described under 'Diagnostic Messages,' below. If your machine includes the timer feature, the listing concludes with a statement of the CPU time taken for the compilation and the elapsed time during the compilation; these times will differ only in a multiprogramming environment.

The following paragraphs describe the optional parts of the listing in the order in which they appear. Appendix A includes a fully annotated example of a compiler listing.

Listings	Option Required
Options for the compilation	OPLIST
Preprocessor input	SOURCE2
Source program	SOURCE
Statement nesting level	NEST
Attribute table	ATR
Cross-reference table	XREF
Aggregate-length table	ATR or XREF
External symbol dictionary	EXTREF
Object module	LIST
Diagnostic messages for severe errors, errors, and warnings	FLAGS, FLAGE, FLAGW

Figure 5-5. Optional Components of Compiler Listing

OPTIONS USED FOR THE COMPILATION

If the option OPLIST applies, a complete list of the options for the compilation, including the default options, follows the statement of the options specified in the EXEC statement. This information appears twice, the second list being in a standard

format to facilitate the automatic collection of operating-system usage statistics.

PREPROCESSOR INPUT

If both the options MACRO and SOURCE2 apply, the compiler lists the input statements to the preprocessor, one record per line. The lines are numbered sequentially at the left.

If the compiler detects an error or the possibility of an error, during the preprocessor phase, it prints a message on the page or pages following the input listing. The format and classification of the error messages are exactly as described for the compilation error messages described under 'Diagnostic Messages,' below.

SOURCE PROGRAM

If the option SOURCE applies, the compiler lists the source program input, one record per line; if the input statements include carriage control characters, the lines will be spaced accordingly. The statements in the source program are numbered sequentially by the compiler, and the number of the first statement in the line appears to the left of each line in which a statement begins. The statements contained within a compound (IF or ON) statement are numbered as well as the compound statement itself; and, when an END statement closes more than one group or block, all the implied END statements are included in the count:

```
 1  P: PROC;
 2  X: BEGIN;
 3     IF A=B
 4        THEN A=1;
 5        ELSE DO;
 6                A=0;
 7                C=B;
 8     END X;
10     D=E;
11     END;
```

If the source statements were generated by the preprocessor, columns 73-80 contain the following information:

Column

73-77 Input line number from which the source statement was generated. This number corresponds to the line number in the preprocessor input listing.

78,79 Two-digit number giving the maximum depth of replacement for this line. If no replacement occurred, the columns are blank.

80 'E' signifies that an error occurred while replacement was being attempted. If no error occurred, the column is blank.

Statement Nesting Level

If the options SOURCE and NEST apply, the block level and the DO level are printed to the right of the statement number under appropriate headings:

```
STMT LEVEL NEST

  1              A: PROC OPTIONS(MAIN);
  2    1            B: PROC(L);
  3    2               DO I=1 TO 10;
  4    2    1            DO J=1 TO 10;
  5    2    2               END;
  6    2    1            BEGIN;
  7    3    1              END;
  8    2    1            END B;
  9    1              END A;
```

ATTRIBUTE AND CROSS-REFERENCE TABLE

If the option ATR applies, the compiler prints an attribute table containing an alphameric list of the identifiers in the program together with their declared and default attributes. If the option XREF applies, the compiler prints a cross-reference table containing an alphameric list of the identifiers in the program together with the numbers of the statements in which they appear. If both ATR and XREF apply, the two tables are combined.

Except for file attributes, the attributes printed will be those that obtain after conflicts have been resolved and defaults applied. Since the file attribute analysis does not take place until after the attribute list has been prepared, the attributes that appear in the list for a file are those supplied by you, regardless of conflicts.

If either of the options ATR and XREF applies, the compiler also prints an aggregate-length table, which gives, where possible, the lengths in bytes of all major structures and all non-structured arrays in the program.

Attribute Table

If an identifier was declared explicitly, the number of the DECLARE statement is listed under the heading DCL NO. The statement numbers of statement labels and entry labels are also given under this heading.

The attributes INTERNAL and REAL are never included; they can be assumed unless the respective conflicting attributes EXTERNAL and COMPLEX appear.

For a file identifier, the attribute EXTERNAL appears if it applies; otherwise, only explicitly declared attributes are listed.

For an array, the dimension attribute is printed first; the bounds are printed as in the array declaration, but expressions are replaced by asterisks.

For a character string or a bit string, the length preceded by the word 'STRING,' is printed as in the declaration, but an expression is replaced by an asterisk.

Cross-Reference Table

If the cross-reference table is combined with the attribute table, the numbers of the statements in which an identifier appears follow the list of attributes for that identifier. The number of a statement in which a based variable identifier appears will be included, not only in the list of statement numbers for that variable, but also in the list of statement numbers for the pointer associated with it.

Aggregate Length Table

Each entry in the aggregate-length table consists of an aggregate identifier preceded by a statement number and followed by the length of the aggregate in bytes.

The statement number is the number either of the DECLARE statement for the aggregate or, for a CONTROLLED aggregate, of an ALLOCATE statement for the aggregate. An entry appears for every ALLOCATE statement involving a CONTROLLED aggregate, since such statements have the effect of changing the length of the aggregate during execution. Allocation of a BASED aggregate does not have this effect, and only one entry, which is that corresponding to the DECLARE statement, appears.

The length of an aggregate may not be known at compilation, either because the aggregate contains elements having adjustable lengths or dimensions, or because the aggregate is dynamically defined. In these cases, the word 'ADJUSTABLE' or 'DEFINED' appears in the LENGTH IN BYTES column.

An entry for a COBOL mapped structure, that is, for a structure into which a COBOL record is read or from which a COBOL record is written, has the word '(COBOL)' appended, but such an entry will appear only if the structure does not consist entirely of:

1. doubleword data, or
2. fullword data, or
3. halfword binary data, or
4. character string data, or
5. aligned bit string data, or
6. a mixture of character string and aligned bit string data.

If a COBOL entry does appear it is additional to the entry for the PL/I mapped version of the structure.

STORAGE REQUIREMENTS

If the option SOURCE applies, the compiler lists the following information under the heading STORAGE REQUIREMENTS on the page following the end of the aggregate-length table:

1. The storage area in bytes for each procedure.
2. The storage area in bytes for each BEGIN block.
3. The storage area in bytes for each ON unit.
4. The length of the program control section (CSECT). The program control section is the part of the object module that contains the executable part of the program.

5. The length of the static internal control section. This control section contains all storage for variables declared STATIC INTERNAL.

TABLE OF OFFSETS

If the options SOURCE, NOSTMT, and NOLIST apply, the compiler lists, for each primary entry point, the offsets at which the various statements occur. This information is found, under the heading TABLE OF OFFSETS AND STATEMENT NUMBERS WITHIN PROCEDURE, following the end of the storage requirements table.

EXTERNAL SYMBOL DICTIONARY

If the option EXTREF applies, the compiler lists the contents of the external symbol dictionary (ESD) for the object module. The ESD is a table containing all the external symbols that appear in the module. (The machine instructions in the object module are grouped in blocks called control sections; an external symbol is a name that can be referred to in a control section other than the one in which it is defined.) The information appears under the following headings:

SYMBOL - An 8-character field that identifies the external symbol.

TYPE - Two characters from the following list to identify the type of ESD entry:

SD - Section definition: the name of a control section within this module.

CM - Common area: a type of control section that contains no executable instructions. The compiler creates a common area for each non-string element variable declared STATIC EXTERNAL without the INITIAL attribute.

ER - External reference: an external symbol that is not defined in this module.

PR - Pseudo-register: a field in a communications area, the pseudo-register vector (PRV), used by the compiler and the library subroutines.

LD - Label definition: the name of an entry point to the external procedure other than that used as the name of the program control section.

ID - Four-digit hexadecimal number: the entries in the ESD are numbered sequentially, commencing from 0001.

ADDR - Hexadecimal representation of the address of the symbol: this field is not used by the compiler, since the address is not known at compile time.

LENGTH - The hexadecimal length in bytes of the control section (SD, CM, and PR entries only).

Standard ESD Entries

The external symbol dictionary always starts with seven standard entries (Figure 5-6):

1. Name of the program control section (the control section that contains the executable instructions of the object module). This name is the first label of the external procedure statement.

2. Name of the static internal control section (which contains storage for all variables declared STATIC INTERNAL). This name is the first label of the external procedure statement, padded on the left with asterisks to seven characters if necessary, and extended on the right with the character A.

3. IHEQINV: pseudo-register for the invocation count (a count of the number of times a block is invoked recursively).

4. IHESADA: entry point of the library routine that obtains automatic storage for a block.

5. IHESADB: entry point of the library routine that obtains automatic storage for variables whose extents are not known at compile time.

6. IHEQERR: pseudo-register used by the library error-handling routines.

7. IHEQTIC: pseudo-register used by the library multitasking routines.

SYMBOL	TYPE	ID	ADDR	LENGTH
FIG5	SD	0001	000000	00033A
***FIG5A	SD	0002	000000	00005F
IHEQINV	PR	0003	000000	000004
IHESADA	ER	0004	000000	
IHESADB	ER	0005	000000	
IHEQERR	PR	0006	000000	000004
IHEQTIC	PR	0007	000000	000004

Figure 5-6. Typical Standard ESD Entries

Other ESD Entries

The remaining entries in the external symbol dictionary vary, but generally include the following:

1. Section definition for the 4-byte control section IHEMAIN, which contains the address of the principal entry point to the external procedure. This control section is present only if the procedure statement includes the option MAIN.

2. Section definition for the control section IHENTRY (always present). Execution of a PL/I program always starts with this control section, which passes control to the appropriate initialization subroutine of the PL/I library; when initialization is complete, control passes to the address stored in the control section IHEMAIN. (Initialization is required only once during the execution of a PL/I program, even if it calls another external procedure; in such a case, control passes directly to the entry point named in the CALL statement, and not to IHENTRY.)

3. LD-type entries for all names of entry points to the external procedure except the first.

4. A PR-type entry for each block in the compilation. The name of each of the pseudo-registers comprises the first label of the external procedure statement, padded on the left with asterisks to seven characters if necessary, and extended on the right with an eighth character selected from one of two tables used by the compiler. If the number of blocks exceeds the number of characters in the first table, the first character of the pseudo-register name is replaced by a character taken from the second table, and the last character recycles. If the first character thus overwritten is the start of the external procedure name rather than an asterisk, the compiler issues a warning message (since identical pseudo-register names could be generated from different procedure names).

   These pseudo-registers are termed display pseudo-registers.

   Example:

```
X: PROC;
   Y: PROC;
      Z: BEGIN;
   END X;
```

   The display pseudo-registers for X, Y, and Z would have the names:

```
******XB
******XC
******XD
```

5. ER-type entries for all the library routines and external routines called by the program. The list includes the names of library routines called directly by compiled code (first-level routines), and the names of routines that are called by the first-level routines.

6. CM-type entries for non-string element variables declared STATIC EXTERNAL without the INITIAL attribute.

7. SD-type entries for all other STATIC EXTERNAL variables and for EXTERNAL file names.

8. PR-type entries for all file names. For EXTERNAL file names, the name of the pseudo-register is the same as the file name; for INTERNAL file names, the compiler generates names as for the display pseudo-registers.

9. PR-type entries for all controlled variables. For external variables, the name of the variable is used for the pseudo-register name; for internal variables, the compiler generates names as for the display pseudo-registers.

STATISTICS

If the option SOURCE applies, the compiler lists the following information after the ESD (or, if the option NOEXTREF applies, after 'Storage Requirements'):

1. Number of records processed by the preprocessor (MACRO records).
2. Number of records processed by the compiler.
3. Number of statements processed by the compiler.
4. Size of object module (in bytes).

OBJECT MODULE

If the option LIST applies, the compiler generates a map of the static internal control section and lists the machine instructions of the object program in a form similar to System/360 assembler language. The machine instructions are described in IBM System/360: Principles of Operation. The following descriptions of the object module listings include many terms that can be properly defined only in the context of an explanation of the mechanism of compilation and the structure of the object program; such an explanation is beyond the scope of this manual.

Both the static internal storage map and the object program listings start on a new page. If the LINECNT option specifies 72 or fewer lines per page and the number of lines to be printed (including skips) exceeds the specified line count, double-column format is used. If the LINECNT option specifies more than 72 lines per page or the number of lines to be printed (including skips) is less than the specified line count, single-column format is used.

Static Internal Storage Map

The first 52 bytes of the static internal control section are of a standard form and are not listed. They contain the following information:

```
DC F'4096'
DC AL4(SI.+X'1000')
DC AL4(SI.+X'2000')
DC AL4(SI.+X'3000')
DC AL4(SI.+X'4000')
DC AL4(SI.+X'5000')
DC AL4(SI.+X'6000')
DC AL4(SI.+X'7000')
DC VL4(IHESADA)
DC VL4(IHESADB)
DC A(DSASUB)
DC A(EPISUB)
DC A(IHESAFA)
```

SI. is the address of the static internal control section, and IHESADA, IHESADB, and IHESAFA are library subroutines. DSASUB and EPISUB are compiler routines for getting and freeing dynamic storage areas (DSAs).

The remainder of the static control section is listed, each line comprising the following elements:

1. Six-digit hexadecimal offset.
2. Up to eight bytes of hexadecimal text.
3. Comment indicating the type of item to which the text refers; a comment appears against the first line only of the text for an item.

The following abbreviations are used for the comments (xxx indicates the presence of an identifier):

DED FOR TEMP or DED	Data element descriptor for a temporary or for a programmer's variable.
FED	Format element descriptor.
DV..xxx	Dope vector for a static variable.
DVD..	Dope vector descriptor.
D.V. SKELETON	Dope vector skeleton for an automatic or controlled variable.
RDV..	Record dope vector.
A..xxx	Address of external control section or entry point, or of an internal label.
ARGUMENT LIST	Argument list skeleton.
CONSTANTS	Constants.
SYMTAB	Symbol table entry.
SYM..xxx	Symbolic name of label constant or label variable.
FILE..xxx	File name.

ON..xxx	Programmer-declared ON-condition.
ATTRIB	File attributes.
xxx	Static variable. If the variable is not initialized, no text appears against the comment, and there is also no static offset if the variable is an array. (This can be calculated from the dope vector if required.)

Object Program Listing

The object program listing includes comments of the following form as an aid to identification of the functions of the components of the program:

* STATEMENT NUMBER n - identifies the start of the code generated for source listing statement number n.
* PROCEDURE xxx - identifies the start of the procedure labeled xxx.
* REAL ENTRY xxx - heads the initialization code for an entry point to a procedure labeled xxx.
* PROLOGUE BASE - identifies the start of the initialization code common to all entry points to that procedure.
* PROCEDURE BASE - identifies the address loaded into the base register for the procedure.
* APPARENT ENTRY xxx - identifies the point of entry into the procedure for the entry point labeled xxx.
* END PROCEDURE xxx - identifies the end of the procedure labeled xxx.
* BEGIN BLOCK xxx - indicates the start of the begin block with label xxx.
* END BLOCK xxx - indicates the end of the begin block with label xxx.
* INITIALIZATION CODE FOR xxx - indicates that the code following performs initial value assignment or dope vector initialization for the variable xxx.

Wherever possible, a mnemonic prefix is used to identify the type of operand in an instruction, and where applicable this is followed by a source program identifier. The following prefixes are used:

A..	Address constant
AE..	Apparent entry point (point in the procedure to which control passed from the prologue).
BLOCK.	Label created for an otherwise unlabeled block (followed by the number of the block).
C..	Constant (followed by a hexadecimal dictionary reference).
CL.	A label generated by the compiler (followed by a decimal number identifying the label).
DED..	Data element descriptor
DV..	Dope vector
DVD..	Dope vector descriptor
FVDED..	Data element descriptor of function value.
FVR..	Function value
IC.	Invocation count pseudo-register.
ON..	ON-condition name
PR..	Pseudo-register
RDV..	Record dope vector
RSW..	Return switch
SI.	Address of static internal control section.
SKDV..	Skeleton dope vector, followed by hexadecimal dictionary reference.
SKPL..	Skeleton parameter list, followed by hexadecimal dictionary reference.
ST..	Symbol table entry
SYM..	Symbolic representation of a label.
TCA..	Temporary control area: a word containing the address of the dope vector of the specified temporary.
TMP..	Temporary, followed by hexadecimal dictionary reference.
TMPDV..	Temporary dope vector, followed by hexadecimal dictionary reference

VO..	Virtual origin
WP1. WP2. WS1. WS2. WS3.	Workspace, followed by decimal number of block which allocates workspace

A listing of the various storage areas is not produced, but the addresses of variables can be deduced from the object program listing.

Example:

A=B+10E1; in the source program produces:

```
0002CA  78 00 B 058  LE    0,B

0002CE  7A 00 B 064  AE    0,C..08F4

0002D2  70 00 B 060  STE   0,A
```

A and B are STATIC INTERNAL variables at an offset of X'60' and X'58', respectively, from the start of the control section.

DIAGNOSTIC MESSAGES

The compiler generates messages that describe any errors or conditions that may lead to error that it detects during compilation. Messages generated by the preprocessor appear in the compiler listing immediately after the listing of the statements processed by the preprocessor; all other messages are grouped together at the end of the listing. The messages are graded according to their severity:

A warning message calls attention to a possible error, although the statement to which it refers is syntactically valid.

An error message describes an attempt made by the compiler to correct an erroneous statement (although it may not specify the corrective action).

A severe error message specifies an error that cannot be corrected by the compiler. The incorrect statement or part of a statement is deleted, but compilation continues. However, if a severe error is detected during the preprocessor stage, compilation is terminated after the compiler has listed the source program.

A termination error message describes an error that forces the termination of the compilation.

The compiler lists only those messages with a severity equal to or greater than that specified by the FLAG option:

Type of Message	Option
warning	FLAGW
error	FLAGE
severe error	FLAGS
termination error	Always listed

Each error message is identified by an 8-character code:

1. The first three characters are IEM, which identify the message as emanating from the F compiler.
2. The next four characters are a 4-digit message number. Appendix K lists all the compiler messages in numeric sequence.
3. The last character is the letter I, which is the operating system code for an informative message.

At the end of a compilation, a message is printed giving the value for the SIZE option that will prevent the spill file being used for dictionary blocks if the program is recompiled.

RETURN CODE

The compiler returns a completion code to the operating system to indicate the degree of success it achieved. This code appears in the job scheduler END OF STEP message as 'RETURN CODE.'

Code	Meaning
0000	No diagnostic messages issued; compilation completed without error; successful execution anticipated.
0004	Warning messages only issued; compilation completed; successful execution probable.
0008	Error messages issued; compilation completed, but with errors; execution may fail.
0012	Severe error messages issued; compilation may have been completed, but with errors; successful execution improbable.
0016	Termination error messages issued; compilation terminated abnormally; successful execution impossible.

Note: This return code is returned for all levels of termination when the syntax check option is used.

Batched Compilation

The batched compilation facility of the compiler allows you to compile more than one external procedure in a single execution of the compiler. The compiler creates an object module for each external procedure and places them sequentially in the data set identified by the DD statement SYSPUNCH or SYSLIN. Batched compilation can increase compiler throughput by reducing operating system overheads, but has the disadvantage that a termination error detected during the compilation of one external procedure will prevent the compilation of those that follow it.

To specify batched compilation, you must include a compiler PROCESS statement in front of each external procedure except the first. This statement indicates to the compiler that it must process another procedure, and it allows you to specify new options for each compilation. The first procedure in the batch does not require a PROCESS statement since the EXEC statement that invokes the compiler contains all the information that it requires.

Note that the return code given for a batched compilation is the highest code that would be returned if the procedures were compiled independently.

THE PROCESS STATEMENT

The format of the PROCESS statement is

```
* PROCESS ('options');
```

where 'options' indicates a list of compiler options exactly as specified in the PARM parameter of an EXEC statement; the list of options must be enclosed within single quotation marks. The asterisk must be in the first byte of the record (card column 1), and the keyword PROCESS may follow in the next byte (column) or after any number of blanks. Blanks are also permitted between:

1. The keyword PROCESS and the option-list delimiter (left parenthesis).
2. The option-list delimiters and the start or finish of the option list.
3. The option-list delimiter and the semicolon.

The options in the option list may include any of those described under 'Optional Facilities,' earlier in this chapter. The options must be separated by commas, and there must be no embedded blanks. The options apply to the compilation of the source statements between the PROCESS statement and the next PROCESS statement. If you omit any of the options, the default values apply; there is no carry over from the preceding EXEC statement or PROCESS statement. The number of characters is limited only by the length of the record. If you do not wish to specify any options, code

```
* PROCESS;
```

The input record that contains the PROCESS statement must be in EBCDIC code.

The OBJNM Option

The OBJNM option determines how the object modules in a batch will be link-edited together. The appearance of this option in the PARM parameter of the EXEC statement or in a PROCESS statement causes the compiler to place a linkage-editor NAME statement at the end of the object module resulting from the compilation of the external procedure to which the option refers. When the batch of object modules is link-edited, the linkage editor places all the modules between one NAME statement and the preceding NAME statement into the same load module; it takes the name of a load module from the NAME statement that follows the last object module that is to be included. For example, consider the following source statements (assuming the option OBJNM=A in the EXEC statement):

```
 ALPHA: PROC OPTIONS(MAIN);
          .
          .
          .
        END ALPHA;
* PROCESS;
 BETA: PROC OPTIONS(MAIN);
          .
          .
          .
        END BETA;
* PROCESS ('OBJNM=B');
 GAMMA: PROC;
          .
          .
          .
        END GAMMA;
```

Compilation of these source statements would result in the following object modules and NAME statements:

```
Object module for ALPHA
  NAME A (R)
Object module for BETA
Object module for GAMMA
  NAME B (R)
```

From this sequence of object modules and control statements, the linkage editor would produce two load modules, one named A containing the object program for procedure ALPHA, and the other named B containing the object programs for the procedures BETA and GAMMA.

You should not specify the OBJNM option if you intend to process the object modules with the linkage loader. The loader processes all object modules with the same name into a single load module; if there is more than one name, the loader recognizes the first one only and ignores the others.

JOB CONTROL LANGUAGE FOR BATCHED PROCESSING

The only special consideration relating to job control statements for batched processing refers to the data set defined by the DD statement SYSLIN. If you include the option LOAD, ensure that this DD statement contains the parameter DISP=(MOD,KEEP) or DISP=(MOD,PASS); the standard cataloged procedures specify DISP=(MOD,PASS). If you do not specify DISP=MOD, successive object modules will overwrite the preceding modules.

Under PCP or MVT, if you do not specify sufficient primary extents for the data sets defined by SYSLIN or SYSPRINT, you may get an abnormal termination with a system completion code of 80A, in which case you should increase the primary extents and run the job again.

Example

Figure 5-7 is an example of a simple batched processing program. It illustrates the use of a single invocation of the cataloged procedure PL1LFCL to compile four procedures and link-edit them into three load modules. Figure 5-8 illustrates how these load modules could later be executed.

The EXEC statement COLE in Figure 5-7 specifies the options for the compilation of the procedure FIRST; of the options specified, only SIZE applies to the compilations of the other procedures. The OBJNM option (abbreviated to 'N') ensures that FIRST will be link-edited into a load module named PGM1, which will contain no other procedures.

The first PROCESS statement requests a listing of the external symbol dictionary for the object module compiled from procedure SECOND. The second PROCESS statement includes the option N=PGM2, which causes the compiler to insert a linkage editor NAME statement at the end of the object module compiled from the procedure PRINT; since this option does not appear in the preceding PROCESS statement, the object modules for procedures SECOND and PRINT will be combined in a single load module (named PGM2) by the linkage editor.

The third PROCESS statement names the load module that will contain the procedure THIRD, and also requests that only error, severe error, and termination error messages be listed by the compiler.

The DD statement LKED.SYSLMOD overrides the corresponding statement in the cataloged procedure, and has the effect of requesting the linkage editor to place the load modules in the private library PUBPGM, from which they can later be called for execution. In Figure 5-8, this library is named again in the DD statement JOBLIB; a library specified by a DD statement of this name serves as an extension of the system program library for the duration of the job in which the statement appears. (Chapters 6 and 12 discuss the linkage editor and program libraries, respectively.)

Compile-time Processing

The compile-time facilites of the (F) compiler are described in IBM System/360 Operating System: PL/I (F) Language Reference Manual. These facilities allow you to include in a PL/I program statements that, when they are executed by the preprocessor stage of the compiler, modify your source statements or cause source statements to be included in your program from a library. The following discussion supplements the information contained in the Language manual by providing some illustrations of the use of the preprocessor and explaining how to establish and use source statement libraries.

```
//J067PGEX JOB
//COLE EXEC PL1LFCL,PARM.PL1L='SIZE=999999,N=PGM1,A',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 FIRST: PROC OPTIONS(MAIN);
         DO I=1250 TO 1500 BY 50;
           DO J=10, 15, 20;
             K=SQRT(I/J);
             PUT SKIP(2) DATA;
         END FIRST;
* PROCESS ('EXTREF');
 SECOND: PROC OPTIONS(MAIN);
         DCL PRINT ENTRY EXT,
             A(5) INIT(1,2,4,8,16),
             B(5) INIT(3,5,7,9,11),
             C(5,5);
         DO I=1 TO 5;
           DO J=1 TO 5;
             C(1,J)=12*A(I)/B(J);
             END;
         END;
         CALL PRINT (A,B,C);
         END SECOND;
* PROCESS ('N=PGM2');
 PRINT:  PROC(THOR,TVERT,ARRAY);
         DCL THOR(*),TVERT(*),ARRAY(*,*);
         I=DIM(THOR,1);
         PUT EDIT (THOR) (X(7), (I) F(7,2));
         DO J=1 TO DIM(TVERT,1);
           PUT SKIP EDIT(TVERT(J), (ARRAY(J,K) DO K=1 TO I))(F(7,2));
         END PRINT;
* PROCESS ('N=PGM3,FE');
 THIRD:  PROC OPTIONS(MAIN);
           ON ENDFILE(SYSIN) GO TO FINISH;
 NEXT:   GET DATA(A,B);
           C=A+8*B**2/3;
           PUT SKIP DATA;
           GO TO NEXT;
 FINISH: END THIRD;
/*
//LKED.SYSLMOD DD UNIT=2311,VOLUME=SER=D186,DSNAME=PUBPGM,DISP=OLD
```

Figure 5-7. An Example of Batched Processing

INVOKING THE PREPROCESSOR

The preprocessor stage of the compiler is executed only if you specify the option MACRO and include a DD statement with the name SYSUT3 in the compilation job step. The compiler uses the data set to hold the preprocessed source statements until compilation begins. The information that you must include in the DD statement is described under 'DD Statements,' earlier in this chapter. The standard cataloged procedures for compilation all include an appropriate DD statement.

The term MACRO owes its origin to the similarity of some applications of the compile-time facilities to the macro language available with such processors as the System/360 assembler. Such a macro language allows you to write a single instruction in your program to represent a sequence of instructions that have previously been defined.

Three other compiler options, MACDCK, SOURCE2, and COMP, are meaningful only when you also specify the MACRO option. All are described earlier in this chapter.

Figure 5-9 is a simple example of the use of the preprocessor to produce a source deck for a procedure SUBFUN; according to the value assigned to the preprocessor variable USE, the source statements will represent either a subroutine or a function.

```
//J067PGE1 JOB
//JOBLIB DD UNIT=2311,VOLUME=SER=D186,DSNAME=PUBPGM,DISP=OLD
//J1 EXEC PGM=PGM1
//SYSPRINT DD SYSOUT=A
//J2 EXEC PGM=PGM2
//SYSPRINT DD SYSOUT=A
//J3 EXEC PGM=PGM3
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
A=27, B=42; A=39, B=17; A=15; B=19; A=12, B=7;
/*
```

Figure 5-8. Execution of the Programs Compiled in Figure 5-7

```
//J068PGEX JOB
//CO EXEC PL1DFC,PARM.PL1D='NOLOAD,NODECK,MACRO,MACDCK,NOCOMP'
//PL1D.SYSIN DD *
 SUBFUN: PROC(CITY);
         DCL IN FILE RECORD,
             1 DATA,
               2 NAME CHAR(10),
               2 POP FIXED(7),
             CITY CHAR(10);
         %DCL USE CHAR;
         %USE='SUB'; /* FOR FUNCTION, SUBSTITUTE %USE='FUN' */
         OPEN FILE(IN);
 NEXT:   READ FILE(IN) INTO(DATA);
         IF NAME=CITY THEN DO;
         CLOSE FILE(IN);
         %IF USE='FUN' %THEN %GO TO L1;
         PUT FILE(SYSPRINT) SKIP LIST(DATA); END;
         %GO TO L2;
 %L1:;   RETURN(POP); END;
 %L2:;   ELSE GO TO NEXT;
         END SUBFUN;
/*
```

Figure 5-9. Using the Preprocessor to Create a Source Deck

THE %INCLUDE STATEMENT

IBM System/360 Operating System: PL/I (F) Language Reference Manual describes how to use the %INCLUDE statement to incorporate source statements from a library into a PL/I source program. (A library is a type of data set that can be used for the storage of other data sets, termed members. Thus, a set of source statements that you may wish to insert into a source program by means of a %INCLUDE statement must exist as a data set (member) within a library. Chapter 12 describes how to create a library and how to place members in it.)

The %INCLUDE statement includes one or more pairs of identifiers. Each pair of identifiers specifies the name of a DD statement that defines a library and, in parentheses, the name of a member of the library. For example, the statement:

```
%INCLUDE DD1(INVERT),DD2(LOOPX)
```

specifies that the source statements in member INVERT of the library defined by the DD statement DD1, and those in member LOOPX of the library defined by DD2, should be inserted into the source program. The compilation job step must include appropriate DD statements.

If you omit the ddname from any pair of identifiers in a %INCLUDE statement, the preprocessor assumes the ddname SYSLIB. In such a case, you must include a DD statement with the name SYSLIB. (Note that the IBM-supplied cataloged procedures do not include a DD statement with this name in the compilation job step.)

Source statements in a library must be in the form of fixed-length records of not more than 100 bytes. The records can be blocked; the maximum blocking factor is 5. The source margin for input records specified by the SORMGIN option applies equally to included statements.

```
//J069PGEX JOB
//STEP1 EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD SYSOUT=A
//SYSUT2 DD DSNAME=NEWLIB,DISP=(NEW,KEEP),UNIT=2311,
//        VOLUME=SER=D186,SPACE=(CYL,(4,,1))
//SYSIN DD *
./       ADD NAME=FUN,LEVEL=00,SOURCE=0
 SUBFUN: PROC(CITY);
         DCL IN FILE RECORD,
             1 DATA,
               2 NAME CHAR(10),
               2 POP FIXED DEC(7),
             CITY CHAR(10);
         OPEN FILE(IN);
 NEXT:   READ FILE(IN) INTO(DATA);
         IF NAME=CITY THEN DO;
           CLOSE FILE(IN);
           RETURN(POP);
           END;
         ELSE GO TO NEXT;
         END SUBFUN;
./       ENDUP
/*
```

Figure 5-10. Placing Source Statements in a New Library

You can use the operating system utility program IEBUPDTE to place source statements in a library. This facility is described in the publication IBM System/360 Operating System, Ultilities.

Examples

Figures 5-10 and 5-11 are simple illustrations of how to place source statements in a library and how to include these statements in a source program.

The program in Figure 5-10 places the source statements of the procedure SUBFUN in a new library. In this example, the source statements will represent a function procedure.

Figure 5-11 illustrates the use of a %INCLUDE statement to include the source statements for SUBFUN in the procedure TEST. The library NEWLIB is defined in the DD statement PL1L.SYSLIB, which is added to the statements of the cataloged procedure PL1LFCLG for this job. Since the source statement library is defined by a statement with the name SYSLIB, the %INCLUDE statement need not include a ddname.

Dynamic Invocation of the Compiler

Note: The following discussion assumes that you are familiar with IBM System/360 assembler language.

You can invoke the (F) compiler from an assembler language program by using one of the macro instructions CALL, LINK, XCTL, or ATTACH. If you use the XCTL macro instruction, you cannot specify any options for the compilation; the default options will apply. However, if you use CALL, LINK, or ATTACH, you can specify:

1. Options for the compilation.
2. Alternative ddnames for the data sets to be used by the compiler.
3. The number of the first page of the compiler listing.

Code the macro instructions as follows:

Name	Operation	Operand
[symbol]	CALL	IEMAA,([optionlist] [,[ddnamelist] [,pagenbr]]),VL
symbol	LINK ATTACH	EP=IEMAA, PARAM=([optionlist] [,[ddnamelist] [,pagenbr]]),VL=1

For a full explanation of these macro instructions, refer to IBM System/360 Operating System: Supervisor and Data Management Macro Instructions.

```
//J069PGE1 JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='MACRO,LOAD,NODECK',PARM.LKED=''
//PL1L.SYSLIB DD UNIT=2311,VOLUME=SER=D186,DSNAME=NEWLIB,DISP=OLD
//PL1L.SYSIN DD *
 TEST:    PROC OPTIONS(MAIN);
          DCL NAME CHAR(10),
              NO FIXED(7);
          ON ENDFILE(SYSIN) GO TO FINISH;
 AGAIN:   GET FILE(SYSIN) LIST(NAME);
          NO=SUBFUN(NAME);
          PUT DATA(NAME,NO);
          GO TO AGAIN;
          %INCLUDE FUN;
 FINISH: END TEST;
/*
//GO.IN DD UNIT=2311,VOLUME=SER=D186,DSNAME=POPLIST,DISP=OLD
//GO.SYSIN DD *
'ABERDEEN'
'DONCASTER'
/*
```

Figure 5-11. Including Source Statements from a Library

The entry-point name IEMAA is the symbolic name of the (F) compiler.

The address parameters are:

'optionlist': the address of a variable-length list of compiler options. The list must begin on a halfword boundary. The first two bytes contain a binary count of the number of bytes in the list (excluding the count field). Options in the list must be separated by commas; the list must not include blanks or zeros.

'ddnamelist': the address of a variable-length list of alternative names for the data sets used by the compiler. The list must begin on a halfword boundary. The first two bytes contain a binary count of the number of bytes in the list (excluding the count field). Each entry in the list must occupy an 8-byte field; the sequence of entries is as follows:

Entry	Alternative Name
1	SYSLIN
2	not applicable
3	not applicable
4	SYSLIB
5	SYSIN
6	SYSPRINT
7	SYSPUNCH
8	SYSUT1
9	not applicable
10	SYSUT3

If a ddname is shorter than eight bytes, fill the field on the right with blanks. If you omit an entry, fill its field with binary zeros; however, you may entirely omit entries at the end of the list.

'pagenbr': the address of a 6-byte field containing the number is to be used as the first page number of the compiler listing. The page number must begin on a halfword boundary, and the first halfword must contain the binary value 4 (the length of the remainder of the field). The other four bytes contain the page number in binary form.

VL or VL=1: specifies that the sign bit in the last word of the parameter list is to be set to 1.

Chapter 6: Linkage Editor and Loader

Introduction

An object module produced by the compiler requires further processing before it is suitable for execution. It must be converted into a load module which can be loaded into main storage and executed. Conversion and execution is performed, in either one or two job steps, by one of two operating system programs, the linkage editor and the linkage loader. This chapter describes these programs and the circumstances in which each can be used to the best advantage. Both programs are fully described in IBM System/360 Operating System: Linkage Editor and Loader.

The two linkage programs require the same kind of input, perform the same basic process (the resolution of external references within the object module), and produce the same result, that is, a load module for execution. They differ in the way they are used and in what they do with the load modules they create.

Linkage loader: Execution by the linkage loader requires one job step, in which a load module is created, loaded into main storage, and executed.

Linkage editor: The linkage editor does not cause the load modules it creates to be loaded and executed. Instead, each load module is placed in a program library; a further job step is required for the loading and execution of such a load module.

CHOICE OF LINKAGE PROGRAM

The two programs are compatible in the following respects:

1. All object modules acceptable as input to a linkage editor are acceptable as input to a linkage loader.
2. All load modules produced by a linkage editor, except those produced with the NE (not editable) attribute are acceptable as input to a linkage loader. (When the NE attribute is produced, the resulting load module has no external symbol dictionary and cannot be reprocessed; the external symbol dictionary is discussed below in the linkage-editor section.)

If you want to keep the load module, or use facilities that are not available to the linkage loader, such as providing an overlay structure, you must use the linkage editor. The linkage loader is essentially a one-shot program checkout facility with limited application.

The differences between the two programs can be summarized as:

Linkage editor:

1. Does not cause the load module to be executed.
2. Can produce more than one load module from a batched compilation.
3. Always places load modules in a library, from which they can be loaded for execution in a later job or job step.
4. Can accept input from other sources as well as the primary input source.
5. Can provide an overlay structure for a program.
6. Can be used to modify existing load modules.

Linkage loader:

1. Requires only one job step for processing, loading, and execution.
2. Can only produce one load module from a batched compilation.
3. Always loads this module into main storage and executes it.
4. The load module exists only for the duration of the job step.
5. Can accept input only from the primary source.
6. Cannot provide an overlay structure for a program, or modify existing load modules.

Performance Considerations

The execution time of a load module is the same whether it is created by the linkage editor or the linkage loader. However, the

editing and loading time for a module is greatly reduced when the linkage loader is used. This is achieved by reductions in:

1. Scheduling time: The object program is processed, loaded, and executed in one job step.

2. Processing time: The linkage loader can process a module in approximately half the time required by the linkage editor, because:

   a. Linkage editor intermediate and I/O operations are eliminated.

   b. The I/O time for reading modules can be reduced by the use of improved buffering techniques and chained scheduling.

3. Amount of auxiliary storage: If the linkage loader input is the object module in a compile-load-and-go job, the auxiliary storage that would be required by the linkage editor intermediate and output data sets is not needed. If the linkage loader input is taken from modules link-edited into a library, the auxiliary storage requirements for the library can be reduced by storing the modules with unresolved library references; these references can be resolved at load time.

Linkage Editor

The linkage editor is an operating system service program that creates load modules. It always places the load modules in a library, from which the job scheduler can call them for execution.

The input to the linkage editor can include object modules, load modules, and control statements that specify how the input should be processed. The output from the linkage editor comprises one or more load modules.

In addition to its primary function of converting object modules into load modules, the linkage editor can also be used to:

- Combine previously link-edited load modules
- Modify existing load modules
- Construct an overlay program

A module constructed as an overlay program can be executed in an area of main storage that is not large enough to contain the entire module at one time. The linkage editor subdivides the module so that it can be loaded and executed segment by segment.

MODULE STRUCTURE

Object and load modules have identical structures. They differ only in that a load module has been processed by the linkage editor and stored in a library with certain descriptive information required by the job scheduler; in particular, the module is marked as 'executable' or 'not executable.' A module comprises three types of information:

- Text (TXT)
- External symbol dictionary (ESD)
- Relocation dictionary (RLD)

Text

The text of an object or load module comprises the machine instructions that represent the program to be executed. These instructions are grouped in blocks termed control sections; a control section is the smallest separately executable unit within a program. An object module created by the PL/I (F) compiler includes the following control sections:

- Control section for the shared library transfer vector. (This is an area used for communication between library modules in the PL/I shared library and those in the partition or region.)
- Program control section: contains the executable part of the program.
- Static internal control section: contains storage for all variables declared STATIC INTERNAL and for constants and static system blocks.
- Control sections termed common areas: one common area is created for each EXTERNAL file name and for each non-string element variable declared STATIC EXTERNAL without the INITIAL attribute.
- IHENTRY: execution of a PL/I program always starts with this control section, which passes control to the appropriate initialization routine; when initialization is complete, control passes to the address stored in the control section IHEMAIN.

- IHEMAIN: for a procedure with the MAIN option, contains the starting address for execution of the PL/I program.
- Control sections for PL/I library modules link-edited with the program.

External Symbol Dictionary

The external symbol dictionary (ESD) contains a list of all the external symbols that appear in the module. An external symbol is a name that can be referred to in a control section other than the one in which it is defined.

The names of the control sections are themselves external symbols, as are the names of variables declared with the EXTERNAL attribute and entry names in the external procedure of a PL/I program. References to external symbols defined elsewhere are also considered to be external symbols; they are known as external references. Such external references in a PL/I object module always include the names of the PL/I subroutine library modules that will be required for the execution of the program. They may also include calls to your own subroutines that are not part of the PL/I subroutine library, nor already included within the object module. Part of the linkage editor's job is to locate the subroutines referred to, and to include them in the load module that will be executed.

Relocation Dictionary

At execution time, the machine instructions in a load module (including the instructions generated by the PL/I (F) compiler) use two methods of addressing locations in main storage:

1. Names used only within a control section have addresses related to the starting point of the control section.
2. Other names (external names) have absolute addresses so that any control section can refer to them.

The relocation dictionary (RLD) contains information that enables the absolute addresses to be established when a module is loaded into main storage for execution. These addresses cannot be determined earlier because the starting address is not known until the module is loaded. The linkage editor consolidates the RLD entries in the input modules into a single relocation dictionary when it creates a load module.

LINKAGE EDITOR PROCESSING

A PL/I compiled program cannot be executed until the appropriate PL/I subroutine library modules have been incorporated. The library modules are included in two ways:

1. By incorporation in the load module during linkage editing.
2. By dynamic call during execution.

The first method is used for most of the PL/I subroutine library modules; the following paragraphs describe how the linkage editor locates the modules. The second is used for modules concerned with input and output (including those used for opening and closing files), and for the modules that issue the execution time error messages. Appendix E lists all the library modules, indicating which are loaded dynamically.

In its basic processing mode, which is illustrated in Figure 6-1, the linkage editor accepts data from its primary input source, a data set defined by a DD statement named SYSLIN. For a PL/I program, this input data is the object module created by the compiler. The linkage editor uses the external symbol dictionary in the input module to determine whether the module includes any external references for which there are no corresponding external symbols in the module: it attempts to resolve such references by a method termed automatic library call.

External symbol resolution by automatic library call involves a search of the library defined by a DD statement named SYSLIB; for a PL/I program this will be the PL/I subroutine library (SYS1.PL1LIB). The linkage editor locates the modules in which the external symbols are defined (if such modules exist), and incorporates them in the load module it is creating.

The linkage editor always places the new load module in the library defined by the DD statement named SYSLMOD.

Any linkage editor processing additional to the basic mode described above must be requested by linkage editor control statements placed in the primary input. These control statements are described at the end of this chapter under 'Additional Processing.'

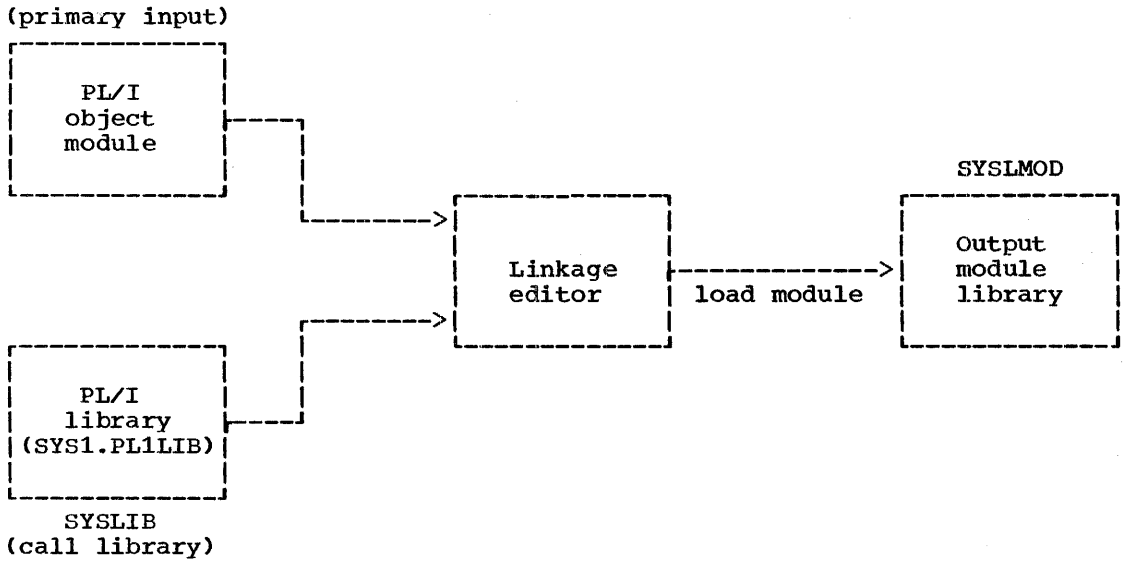


Figure 6-1. Basic Linkage Editor Processing

Main Storage Requirements

Two levels of the linkage editor are currently available; each has a number of different versions. The E-level is available in 15K, 18K, and 20K versions; the F-level is available in 44K, 88K, and 128K versions. The capabilities and capacities of each version are described in System/360 Operating System: Linkage Editor and Loader.

Job Control Language for Link-Editing

Although you will probably use cataloged procedures rather than supply all the job control statements required for a job step that invokes the linkage editor, it is necessary to be familiar with these statements so that you can make the best use of the linkage editor and, if necessary, modify the statements of the cataloged procedures.

The IBM-supplied PL/I cataloged procedures that refer to the linkage editor are:

PL1LFCL	Compile and link-edit
PL1LFCLG	Compile, link-edit, execute
PL1LFLG	Link-edit and execute

Chapter 8 discusses these cataloged procedures and describes how to modify or override the statements they contain.

The following paragraphs describe the essential job control statements for link-editing; they use statements from the PL/I cataloged procedures as typical examples. Appendix B contains a description of the parameters of the DD statements that are referred to.

EXEC STATEMENT

The name IEWL is an alias for the linkage-editor program. If you use the name IEWL in the EXEC statement that invokes the linkage editor, the job scheduler will load the version to which this name corresponds. Normally this would be either the one which is the largest available within your operating system, or the one which is the most suitable for your job. You should consult your systems programmer if you need to know what versions of the linkage editor are available at your installation, and how to invoke them. The basic EXEC statement is:

```
// EXEC PGM=IEWL
```

By using the PARM parameter of the EXEC statement, you can select one or more of the optional facilities offered by the linkage editor; these facilities are discussed under 'Optional Facilities,' below. The use of the other parameters of the EXEC statement is as described in Chapter 7, 'Executing the Load Module.'

ddname	Function	Possible device classes[1]
SYSLIN	Primary input data, normally the output from the compiler	SYSSQ or the input job stream (specified by DD *)
SYSLMOD	Output for load module	SYSDA
SYSUT1	Additional workspace	SYSDA
SYSPRINT	Listing, including diagnostic messages	SYSSQ or SYSOUT device
SYSLIB	Automatic call library (usually the PL/I subroutine library)	SYSDA

[1]SYSSQ Magnetic-tape or direct-access device
SYSDA Direct-access device

Figure 6-2. Linkage-Editor Data Sets

DD STATEMENTS

The linkage editor always requires four standard data sets. You must define these data sets in DD statements with the standard names SYSLIN, SYSLMOD, SYSUT1, and SYSPRINT.

A fifth data set, defined by a DD statement with the name SYSLIB, is necessary if you want to use the automatic library call facility. The five standard data sets are summarized in Figure 6-2.

Primary Input (SYSLIN)

The primary input source must be a CONSECUTIVE data set containing one or more object modules and/or linkage-editor control statements; a load module cannot be part of the primary input, although it can be introduced by the control statement INCLUDE. For a PL/I program, the primary input source is usually a data set containing an object module created by the compiler. The data set may be on magnetic-tape or on a direct-access device, or you can include it in the input job stream. In all cases, the input must be in the form of 80-byte F-format records.

The cataloged procedure PL1LFLG includes the DD statement:

```
//SYSLIN DD DDNAME=SYSIN
```

This statement specifies that the primary input data set must be defined in a DD statement named SYSIN. If you use this catalogue procedure, you must supply this DD statement, specifying the qualified ddname LKED.SYSIN. For example, to link-edit and execute a PL/I object module placed in the input stream, you can use the following statements:

```
//LEGO JOB
// EXEC PL1LFLG
//LKED.SYSIN DD *

Insert here the object module to be
  link edited and executed

/*
```

Note:

1. If modules with identical names appear in the primary input, the linkage editor processes only the first of them.
2. You can include load modules or object modules from one or more libraries in the primary input by using a linkage editor INCLUDE statement; refer to 'Additional Processing,' below.

Output (SYSLMOD)

The linkage editor always places the load modules that it creates in a library defined by a DD statement with the name SYSLMOD. (A library is a type of direct-access data set that can be used for the storage of other consecutive data sets, frequently load modules); the data sets stored in a library are termed members. To store a member in a library, include the parameter DSNAME=dsname(membername) in the DD statement that defines the library; replace 'dsname' with the name of the library, and 'membername' with the name of the member.)

The PL/I cataloged procedures include the following DD statement:

```
//SYSLMOD DD DSNAME=&&GOSET(GO),
//           DISP=(MOD,PASS),
//           UNIT=SYSDA,
//           SPACE=(1024,
//           (50,20,1),RLSE)
```

This statement defines a temporary library named &&GOSET and assigns the name GO to the load module that the linkage editor will place in it. If you want to retain the load module after execution of your job, you must replace this DD statement with one that defines your own permanent library. For example, assume that you have a library called USLIB on 2311 disk pack serial number 371; to name the load module MOD1 and place it in this library, code:

```
//LKED.SYSLMOD DD DSNAME=USLIB(MOD1),
//           UNIT=2311,
//           VOLUME=SER=371,
//           DISP=OLD
```

The size of a load module must not exceed 512K bytes (512 * 1024 bytes) for programs executed under PCP or MFT; a much larger load module is permitted for MVT. The SPACE parameter in the DD statement SYSLMOD used in the PL/I cataloged procedures allows for an initial allocation of 50K bytes and, if necessary, 15 further allocations of 20K bytes (a total of 350K bytes); this should suffice for most applications.

Workspace (SYSUT1)

The linkage editor requires a temporary data set on a direct-access device for use as extra workspace. The DD statement that defines this data set must have the name SYSUT1. The following statement contains the essential parameters:

```
//SYSUT1 DD UNIT=SYSDA,
//          SPACE=(1024,(200,20))
```

You should never need to modify the DD statement SYSUT1 in a cataloged procedure.

If your installation supports dedicated workfiles, these can be used to provide workspace for the link-edit job step. For details of these workfiles and their use, see 'Workspace (SYSUT1,SYSUT3)' in Chapter 5, 'Compilation.'

Listing (SYSPRINT)

The linkage editor generates a listing that includes reference tables relating to the load modules that it produces and also, when necessary, diagnostic messages. The information that may appear is described under 'Listing,' below.

You must define the data set on which you wish the linkage editor to place its listing in a DD statement named SYSPRINT. The data set must have CONSECUTIVE organization. Although the listing is usually printed, it can be written on any type of magnetic-tape or direct-access device. For printed output, the following statement will suffice:

```
//SYSPRINT DD SYSOUT=A
```

Automatic Call Library (SYSLIB)

If you want the linkage editor to resolve external references by automatic library call, you must use a DD statement with the name SYSLIB to define the library which the linkage editor must search. You can cause the linkage editor to search more than one library by concatenating the DD statements that define the libraries: include the ddname SYSLIB in the first statement and leave the name fields of the following statements blank.

The link-editing of a PL/I object module normally requires the presence of a SYSLIB statement that refers to the PL/I subroutine library (SYS1.PL1LIB).

The automatic call library can contain load modules or object modules, but not both.

EXAMPLE

The following example is a typical sequence of job control statements for link-editing a PL/I object module. The DD statement SYSLIN indicates that the object module will follow immediately in the input stream; for example, it might be an object deck created by invoking the PL/I (F) compiler with the DECK option (see Chapter 5). The DD statement SYSLMOD specifies that the linkage editor should name the new load module LKEX, and that it should place it in a new library named MODLIB; the presence of the SPACE parameter and the keyword NEW in the DISP parameter indicates to the operating system that this DD

statement requests the creation of a new library.

```
//LINK JOB
// EXEC PGM=IEWL
//SYSLMOD DD UNIT=2311,
//           VOLUME=SER=D186,
//           DSNAME=MODLIB(LKEX),
//           DISP=(NEW,KEEP),
//           SPACE=(CYL,(10,10,1))
//SYSUT1 DD UNIT=2311,
//           SPACE=(1024,(200,20))
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSNAME=SYS1.PL1LIB,
//           DISP=OLD
//SYSLIN DD *

Insert here the object module to be
link-edited

/*
```

Optional Facilities

The linkage editor provides a number of optional facilities that you can select by including the appropriate keywords from the following list in the PARM parameter of the EXEC statement that invokes it:

```
LIST
MAP or XREF
LET or XCAL
NCAL
SIZE
```

The PARM parameter is a keyword parameter. Code PARM= followed by the list of options, separating the names of the options with commas and enclosing the list within single quotation marks. For example:

```
// EXEC PGM=IEWL,PARM='LIST,MAP'
```

If you are using a cataloged procedure, you must include the PARM parameter in the EXEC statement that invokes the procedure and qualify the keyword PARM with the name of the procedure step that invokes the linkage editor. For example:

```
// EXEC PL1LFCLG,PARM.LKED='LIST,XREF'
```

The following paragraphs describe the optional facilities. The listing produced by the options LIST, MAP, and XREF are described under 'Listing,' below.

LIST

The LIST option specifies that all linkage editor control statements processed should be listed in the data set defined by the DD statement SYSPRINT.

MAP

The MAP option requests the linkage editor to produce a map of the load module; this map indicates the relative locations and lengths of the control sections in the module.

XREF

The XREF option requests the linkage editor to produce a map of the load module and a cross-reference list of all the external references in each control section. XREF includes MAP.

LET

The LET option requests the linkage editor to mark the load module 'executable,' even if slight errors or abnormal conditions are found during link-editing.

XCAL

The XCAL option requests the linkage editor to mark the load module as executable even if errors or abnormal conditions, including improper branches between control sections, are found during link-editing. XCAL, which includes LET, applies only to an overlay module.

NCAL

The NCAL option specifies that no external references should be resolved by automatic library call. However, the load module is marked 'executable' (providing there are no errors).

You can use the NCAL option to conserve storage space in private libraries since, by preventing the resolution of external references during link-editing, you can

store PL/I load modules without the relevant PL/I library subroutines; the DD statement SYSLIB is not required. Before executing such load modules, you must link-edit them again to resolve the external references, but the load module thus created need exist only while it is being executed. You can use this technique to combine separately compiled PL/I routines in a single load module.

SIZE

The SIZE option specifies the amount of main storage, in bytes, to be allocated to the linkage editor; it applies only to the F-level linkage editor. Code the SIZE option as a keyword parameter with the following format:

```
SIZE=(value1,value2)
```

For 'value1' substitute a decimal integer number representing the number of bytes of main storage to be allocated to the linkage editor, including the allocation for the load module buffer specified in value2.

For 'value2' substitute a decimal integer number representing the number of bytes of main storage to be allocated for the load module buffer. The linkage editor uses the load module buffer when it reads in load module records, and also to retain information for subsequent writing of output records.

You can specify both values as the actual number of bytes (for example, SIZE=(45056,6144)) or as a multiple of 1024 bytes (for example, SIZE=(44K,6K)). Value1 must exceed value2; the following table lists the minimum and maximum values and the minimum difference between value1 and value2 for the three designs of the F-level linkage editor:

Linkage Editor	Value1		Value2		Value1 - Value2
	Min	Max	Min	Max	(Min)
44K	44K	-	6K	100K	38K
88K	88K	-	6K	100K	44K
128K	128K	-	6K	100K	66K

If you specify SIZE incorrectly, or if you omit the parameter, a default value set at system generation is used.

Listing

The linkage editor always lists, in the data set defined by the DD statement SYSPRINT, any errors or abnormal conditions that it discovers. It will also list the following additional information if you specify the appropriate options:

Listings	Option Required
Control statements processed by the linkage editor	LIST
Map of the load module	MAP or XREF
Cross-reference table of external references	XREF

The following paragraphs describe the elements of the listing; the sequence in which they appear differs between the E-level and F-level linkage editors. Appendix A includes a fully annotated example of a linkage editor listing.

A statement of the level of linkage editor and the options specified appears at the head of the listing.

CONTROL STATEMENTS AND ERRORS

During processing, the linkage editor notes the occurrence of error and other conditions as it encounters them; these notes appear as a list immediately after the heading statement. If you specify the LIST option, this list also includes all control statements processed by the linkage editor.

Each entry in the list comprises a 7-character code followed by the name of the control statement to which the code applies. For a control statement, the code is always IEW0000. All other codes refer to explanatory, error, or warning messages; each code comprises:

1. The letters IEW, which identify linkage editor messages.
2. A 3-digit message number.
3. A 1-digit severity code as follows:

Code	Meaning
0	A condition which will not cause an error during execution. The output module is marked 'executable.'
1	A condition that may cause an error during execution. The output module is marked 'executable.'
2	An error that could make execution impossible. The output module is marked 'not executable' unless LET is specified.
3	An error that will make execution impossible. The output module is marked 'not executable.'
4	An error condition from which no recovery is possible. Linkage editor processing is terminated, and no output other than diagnostic messages is produced.

At the end of the list of messages, the E-level linkage editor places a statement of the disposition of the load module; the F-level linkage editor places this statement at the end of the listing, just before the Diagnostic Message Directory (see below). The disposition statements, with one exception, are self-explanatory; the exception is:

****modulename DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET

The message normally appears when a request has been made for the linkage editor to substitute the new load module for an existing module. It indicates that the linkage editor was unable to locate the existing module, but has placed the new module in the data set named in the DD statement SYSLMOD. If you name a new module by including a name in the DSNAME parameter of the DD statement SYSLMOD, the linkage editor assumes that you want to replace an existing module (even if the data set is new).

DIAGNOSTIC MESSAGE DIRECTORY

When processing of a load module has been completed, the linkage editor lists in full all the diagnostic messages whose numbers appear in the preceding list. IBM System/360 Operating System: Linkage Editor and Loader contains explanations of all the linkage editor messages and their probable causes, and suggests how to cope with them.

The warning message IEW0461 frequently appears in the linkage editor listing for a PL/I program. It refers to external references that have not been resolved because NCAL was specified (in this instance, in a linkage editor LIBRARY statement). The references occur in PL/I library subroutines that are link-edited with your program as a result of automatic library call. Some library modules may, in turn, call other library modules. Any library module that calls a secondary module that may only occasionally be required is preceded by a LIBRARY statement. This specifies that the references to the secondary modules should not be resolved unless these modules are already part of the input to the new load module, that is, they are external references. For those secondary modules that are required, the compiler generates another external symbol dictionary containing alternative names for the modules. These new references can be resolved, and the required modules are placed in the new load module. If the secondary modules in turn call other modules, the process is repeated.

MODULE MAP

The linkage editor listing includes a module map only if you specify the options MAP or XREF. The map lists all the control sections in the output module and all the entry names in each control section. The control sections are listed in order of appearance in the load module; alongside each control section name is its address relative to the start of the load module (address 0) and its length in bytes. The entry points within the load module appear on the printed listing below and to the right of the control sections in which they are defined; each entry point name is accompanied by its address relative to the start of the load module.

Each control section that is included by automatic library call is indicated by an asterisk. For an overlay module, the control sections are arranged by segment in the order in which they were specified.

After the control sections, the module map lists the pseudo-registers established by the compiler. Pseudo-registers are fields in a communications area, the pseudo-register vector (PRV), used by the PL/I library subroutines and compiled code during execution of a PL/I program. The storage occupied by the PRV is not allocated until the start of execution of a PL/I program; therefore, it does not form part of the load module. The addresses

given in the list of pseudo-registers are relative to the start of the PRV.

At the end of the module map, the linkage editor supplies the following information:

1. The length of the PRV.
2. The relative address of the instruction with which execution of the load module will commence (ENTRY ADDRESS).
3. The total length of the module. For an overlay module, the length is that of the longest path.

All the addresses and lengths given in the module map and associated information are in hexadecimal form.

CROSS-REFERENCE TABLE

The linkage editor listing includes a cross-reference table only if you specify the option XREF. This option produces a listing that comprises all the information described under 'Module Map,' above, together with a cross-reference table of external references. The table lists the location of each reference within the load module, the symbol to which the reference refers, and the name of the control section in which the symbol is defined.

For an overlay module, a cross-reference table is provided for each segment. It includes the number of the segment in which each symbol is defined.

Unresolved symbols are identified in the cross-reference table by the entry $UNRESOLVED. However, if a symbol was not resolved owing to the use of the NCAL option or a LIBRARY statement, it is identified by $NEVER-CALL.

RETURN CODE

The linkage editor returns a completion code to the operating system to indicate the degree of success it achieved. This code appears in the job scheduler END OF STEP message as 'RETURN CODE.' The code is derived by multiplying the highest diagnostic message severity code by four.

Code	Meaning
0000	Normal completion
0004	Warning, but execution should be successful
0008	Errors, execution may fail
0012	Severe errors, execution impossible
0016	Termination error

The code 0004 almost invariably appears after a PL/I program has been link-edited, because some external references in the PL/I library subroutines have not been resolved. (Refer to 'Diagnostic Message Directory,' above.)

Additional Processing

The basic function of the linkage editor is to create a single load module from the data that it reads from its primary input source, but it has several other facilities that you can call upon by using linkage editor control statements. The use of those statements of particular relevance to a PL/I program is described under functional headings, below. All the linkage editor control statements are fully described in IBM System/360 Operating System: Linkage Editor and Loader.

FORMAT OF CONTROL STATEMENTS

A linkage editor control statement is an 80-byte record that contains two fields. The operation field specifies the operation required of the linkage editor; it must be preceded and followed by at least one blank character. The operand field names the control sections, data sets, or modules that are to be processed, and it may contain symbols to indicate the manner of processing; the field consists of one or more parameters separated by commas. Some control statements may have multiple operand fields separated by commas.

The position of a control statement in the linkage editor input depends on its function.

In the following descriptions of the control statements, items within brackets [ ] are optional; you may omit them at your discretion.

MODULE NAME

A load module must have a name so that the linkage editor and the job scheduler can identify it. A name comprises up to seven characters, the first of which must be alphabetic.

You can name a load module in two ways:

1. If you are creating a single load module, it is sufficient to include its name as a member in the DSNAME parameter of the DD statement SYSLMOD.

2. If you are creating two or more load modules in a single execution of the linkage editor, you will need to use the NAME statement. (The PL/I (F) compiler can supply the NAME statements when you use the batch-compilation feature; see Chapter 5.)

The format of the NAME statement is:

```
NAME membername [(R)]
```

For 'membername' substitute the name of the module. (R), if present, signifies that the load module is to replace an existing load module of the same name in the data set defined by the DD statement SYSLMOD.

The NAME statement must appear in the primary input to the linkage editor (the data set defined by the DD statement SYSLIN); if it appears elsewhere, the linkage editor ignores it. The statement must follow immediately after the last input module that will form part of the load module it names (or after the INCLUDE statement that specifies the last module.)

Alternative Names

You can use the ALIAS statement to give a load module an alternative name; a load module can have as many as sixteen aliases in addition to the name given to it in a SYSLMOD DD statement or by a NAME statement.

The format of the ALIAS statement is:

```
ALIAS symbol
```

For 'symbol' substitute any name of up to seven characters; the first character must be alphabetic. You can include more than one name in an ALIAS statement; separate the names by commas, for example:

```
ALIAS FEE,FIE,FOE,FUM
```

An ALIAS statement can be placed before, between, or after the modules or other control statements that are being processed to form a new load module, but it must precede the NAME statement that specifies the primary name of the new module.

To execute a load module, you can include an alias instead of the primary name in the PGM parameter of an EXEC statement. Providing the alias is not also the name of an entry point within the module, execution will commence at the normal entry point (which, for a PL/I program, is the control section IHENTRY). Do not use a NAME or an ALIAS statement to give a PL/I load module a name that is an entry name other than IHENTRY. If you do, the initialization routines which are called from IHENTRY before control is passed to your program will be bypassed and your program will not execute successfully. Generally, you should not give a PL/I module a name or an alias name that begins with 'I' (except IHENTRY).

ADDITIONAL INPUT SOURCES

The linkage editor can accept input from sources other than the primary input defined in the DD statement SYSLIN. For instance, the automatic library call facility enables the linkage editor to include modules from the library named in the DD statement SYSLIB. You can name additional input sources by means of the INCLUDE statement, and you can direct the automatic call mechanism to alternative libraries by means of the LIBRARY statement.

INCLUDE Statement

The INCLUDE statement causes the linkage editor to process the module or modules indicated. After the included modules have been processed, the linkage editor continues with the next item in the primary input. If an included sequential data set also contains an INCLUDE statement, that statement is processed as if it were the last item in the data set (Figure 6-3).

The format of an INCLUDE statement is:

```
INCLUDE ddname [(membername)]
```

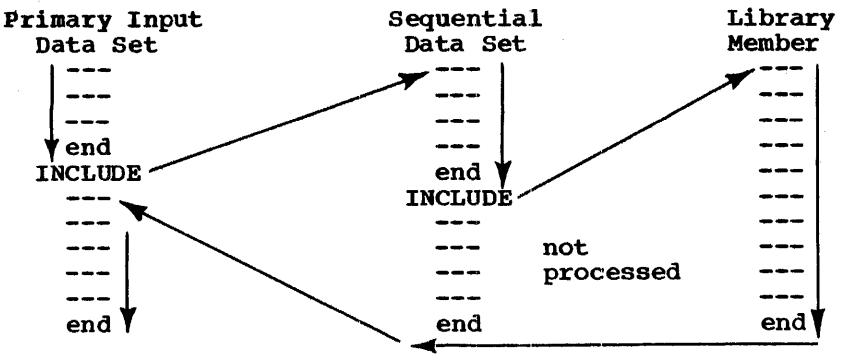


Figure 6-3. Processing of Additional Data Sources

Replace 'ddname' with the name of a DD statement that defines either a sequential data set or a library that contains the modules and control statements to be processed. If the DD statement defines a library, replace 'membername' with the names of the modules to be processed, separated by commas. You can specify more than one ddname, each of which may be followed by any number of member names in a single INCLUDE statement. For example, the statement

```
INCLUDE D1(MEM1,MEM2),D2(MODA,MODB)
```

requests the inclusion of the members MEM1 and MEM2 from the library defined by the DD statement D1, and the members MODA and MODB from the library defined by D2.

LIBRARY Statement

The basic function of the LIBRARY statement is to name call libraries in addition to those named in the DD statement SYSLIB. For this purpose, the format of the statement is similar to that of the INCLUDE statement:

```
LIBRARY ddname(membername)
```

Replace 'ddname' with the name of a DD statement that defines the additional call library, and 'membername' with the names of the modules to be examined by the automatic call mechanism; separate the module names with commas.

You can also use the LIBRARY statement to specify external references that should not be resolved, or to specify that no external references should be resolved. (Refer to IBM System/360 Operating System: Linkage Editor and Loader.)

OVERLAY PROGRAMS

To reduce the amount of main storage required for the execution of a program, you can organize it into an overlay structure. An overlay program is divided into segments, which can be loaded and executed successively in the same area of main storage. To construct such a program, you must use linkage editor control statements to specify the relationship between the segments. Note that one segment, termed the root segment must remain in main storage throughout the execution of the program.

Designing the Overlay Structure

Before preparing the linkage editor control statements, you must determine the overlay tree structure from the program. A tree is a graphic representation that shows which segments occupy main storage at different times. The design of tree structures is discussed in IBM System/360 Operating System: Linkage Editor and Loader, but, for the purposes of this chapter, Figures 6-4 and 6-5 contain a simple example.

The tree in Figure 6-5 represents the execution of the PL/I program of Figure 6-4. The program comprises six procedures, A, B, C, D, E, and F. The main procedure A calls procedures B and F. Procedure B calls procedure C, which, in turn, calls procedures D and E. (Note that only procedure A requires the option MAIN.)

The main procedure (A) must be in main storage throughout the execution of the program. Since the execution of procedure B will be completed before procedure F is called, the two procedures can occupy the same storage; this is depicted by the lines representing the two procedures in Figure 6-5 starting from the common point (node)

```
A: PROC OPTIONS(MAIN);
     .
     .
     CALL B;
     .
     .
     CALL F;
     .
     .
     END A;
```

```
B: PROC;
     .
     .
     CALL C;
     .
     .
     END B;
```

```
C: PROC;
     .
     .
     CALL D;
     .
     .
     CALL E;
     .
     .
     END C;
```

```
D: PROC;
     .
     .
     .
     .
     END D;
```

```
E: PROC;
     .
     .
     .
     .
     END E;
```

```
F: PROC;
     .
     .
     .
     .
     END F;
```

Figure 6-4. Program Suitable for Overlay Structure

X. Procedure B must remain in storage while procedures C, D, and E are executed, but procedures D and E can occupy the same storage; thus the lines representing procedures D and E start from the node Y.

The degree of segmentation that can be achieved can be clearly seen from the diagram. Since procedure A must always be present, it must be included in the root segment. Procedures F, D, and E can usefully be placed in individual segments, as can procedures B and C together; there is nothing to be gained by separating procedures B and C, since they must be present together at some time during execution.

Control Statements

To specify to the linkage editor how you want a load module structured, use the control statements INSERT and OVERLAY. You must include the attribute OVLY in the PARM parameter of the EXEC statement that invokes the linkage editor; if you omit this attribute, the linkage editor will ignore the control statements.

The OVERLAY statement indicates the start of an overlay segment. Its format is:

```
OVERLAY symbol
```

Replace 'symbol' with an arbitrary symbol representing the node at which the segment starts (for example, X in Figure 6-5). You must specify the symbolic origin of every segment, except the root segment, in an OVERLAY statement.

An INSERT statement positions control sections in an overlay segment. Its format is:

```
INSERT control-section-name
```

Replace 'control-section-name' with the names of the control sections (that is, procedures) that are to be placed in the segment; if you include two or more names in the statement, separate them with commas. The INSERT statements that name the control sections in the root segment must precede the first OVERLAY statement.

Creating an Overlay Structure

The most efficient method of defining an overlay structure, and the simplest for a PL/I program, is to group all the OVERLAY and INSERT statements together and then place them in the linkage editor input (SYSIN) after the modules that form the

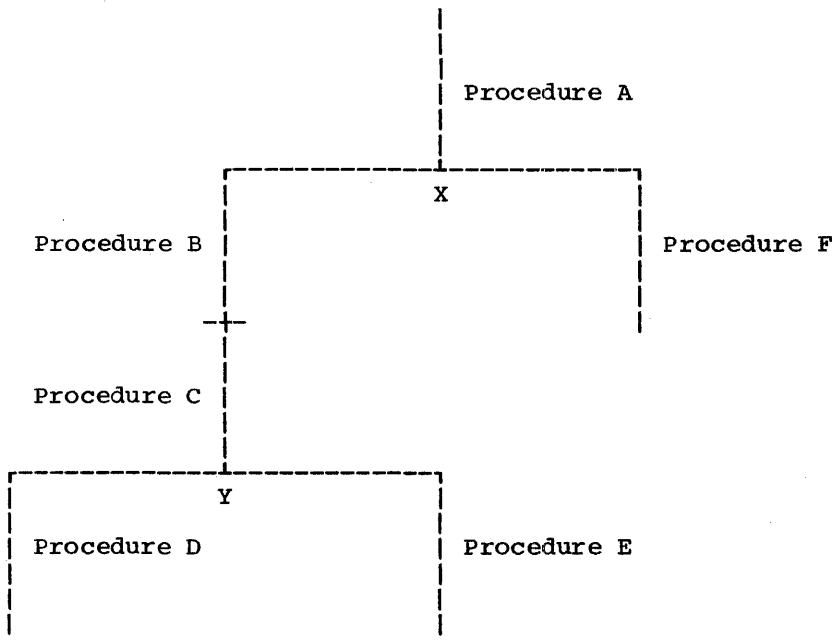


Figure 6-5. Overlay Tree Structure for Program of Figure 6-4

program. The linkage editor initially places all the input modules in the root segment, and then moves those control sections that are referred to in INSERT statements into other segments.

This method has the advantage that you can use the batch compilation facility of the compiler to process all the procedures in one run and place the object modules in a temporary CONSECUTIVE data set. You can then place the linkage editor control statements in the input stream, concatenating them with the data set that contains the object modules. (Do not use the compiler OBJNM option to name the object modules; if you do, the NAME statements inserted by the compiler will cause the linkage editor to attempt to create separate load modules rather than a single overlay module.)

Figure 6-6 illustrates how you could use the PL/I cataloged procedure PL1LFCLG to create and execute the overlay structure of Figure 6-5.

An alternative approach instead of batch compilation is to compile the procedures independently and store them as load or object modules in a private library. You can then use an INCLUDE statement to place them in the input to the linkage editor (SYSLIN).

```
//CREX JOB
// EXEC PL1LFCLG,PARM.LKED='OVLY'
//PL1L.SYSIN DD *

Source statements for procedure A

*PROCESS

Source statements for procedure B

*PROCESS

Source statements for procedure C

*PROCESS

Source statements for procedure D

*PROCESS

Source statements for procedure E

*PROCESS

Source statements for procedure F
/*
//LKED.SYSIN DD *
 OVERLAY X
 INSERT B,C
 OVERLAY Y
 INSERT D
 OVERLAY Y
 INSERT E
 OVERLAY X
 INSERT F
/*
```

Figure 6-6. Compiling, Link-Editing, and Executing an Overlay Program

If an INSERT statement contains the name of an external procedure, the linkage editor will move only the related program control section, which has the same name. All other control sections established by the compiler, and all the PL/I library subroutines, will remain in the root segment.

It is important that PL/I library subroutines be in the root segment, since the PL/I (F) compiler does not support exclusive calls (calls between segments that do not lie in the same path). For example, in the structure of Figure 6-5, procedures in the segment containing D; could call procedures in the segments containing A, B, C, and D, but not in the segments containing E or F. Procedures in the segments containing B and C could call procedures in the segments containing A, B, C, D, and E, but not in the segment containing F. A procedure in the segment containing B may not call a procedure in the segment containing A if this latter procedure calls a procedure in the segment containing F.

Note: The library modules IHETABS and IHEMAIN must be in the root segment.

However, certain modules may not be required by all segments, in which case you can move them into a lower segment. To do this, compile the procedures using the compiler option EXTREF, and then examine the external symbol dictionary. For example, if in the structure of Figure 6-5 the module IHESNS is called only by the segment containing E, you can move into that segment by placing the control statement INSERT IHESNS immediately after the statement INSERT E.

Similarly, you can move data control sections from the root segment to lower segments. For example, to move the static internal control section for procedure F into the segment containing F, place the statement INSERT ******FA after the statement INSERT F. Note that values assigned to static data items are not retained when a segment is overlaid. (A storage area in static constitutes static data for this purpose, but still has the same use as a DSA). Therefore, do not move static data from the root segment unless it comprises only:

1. Values set by the INITIAL attribute and then unchanged (i.e., read-only data).

2. Values that need not be retained between different loadings of the segment.

An alternative method of creating an overlay structure is to obtain object decks for the procedures that form the program, and then to intersperse OVERLAY statements among them in the linkage editor input. This method requires more care, since you must move into the root segment all static internal control sections (unless they are read-only) and control sections that refer to external variables not included in a common area. The linkage editor automatically places common areas and any library subroutines that are used in common by different procedures, in the common segments of the paths in which they are referred to. For example, if only procedures D and E of Figure 6-5 require the subroutine IHEOST, the linkage editor will place it in the segment that contains procedures B and C; but if procedure F also refers to IHEOST, the linkage editor will place it in the root segment.

Linkage Loader

The linkage loader is an operating system program that creates and executes load modules. The modules created are always placed directly into main storage (never in a library) and executed.

The input to the linkage loader is a single object or load module or several object or load modules, or a mixture of both types. The output is always a single load module in main storage.

The linkage loader does not support the linkage editor control statements (ALIAS, CHANGE, ENTRY, INCLUDE, INSERT, LIBRARY, NAME, OVERLAY, REPLACE, SETSSI). The presence of any of these in the job stream will not be treated as an error; the job will continue to be processed, and the name of the statement is printed on SYSLOUT together with a diagnostic message.

The linkage loader compensates for the absence of the facilities provided by these control statements by allowing the concatenation of both object and load modules in the data set defined by the ddname SYSLIN, and by allowing an entry point to be specified in the EP option of the PARM parameter (see below in 'Optional Facilities').

MODULE STRUCTURE

The structure of a module which is the input to the linkage loader is the same as that for a module which is the input to the

linkage editor. This structure has already been described in 'Module Structure,' in the linkage editor section.

LINKAGE LOADER PROCESSING

The linkage loader processes the input module or modules in order to resolve all external references in control sections. Once this has been accomplished, the load module is loaded into main storage and executed. The basic functions are:

1. Resolution of external references between control sections in program modules.
2. Resolution of other external references by inclusion of modules from the PL/I subroutines library (situated either on a direct-access device or in main storage).
3. Automatic editing by deleting duplicate copies of program modules.
4. Using the relocation dictionary (RLD) to obtain absolute addresses for control sections; if a particular control section is not in main storage when required, the RLD is saved until the control section has been loaded.

In its basic processing mode, which is illustrated in Figure 6-7, the linkage loader accepts data from its primary input source, a data set defined by a DD statement named SYSLIN. For a PL/I program, this input data is the object module created by the compiler. The linkage loader uses the external symbol dictionary in the input module to determine whether the module includes any external references for which there are no corresponding external symbols in the module: it attempts to resolve such references by a method termed automatic library call.

External symbol resolution by automatic library call involves a search of the library defined by a DD statement named SYSLIB; for a PL/I program this will be the PL/I subroutine library (SYS1.PL1LIB). The linkage loader locates the modules in which the external symbols are defined (if such modules exist), and incorporates them in the load module it is creating. If all the external references have been resolved satisfactorily, the load module is executed.

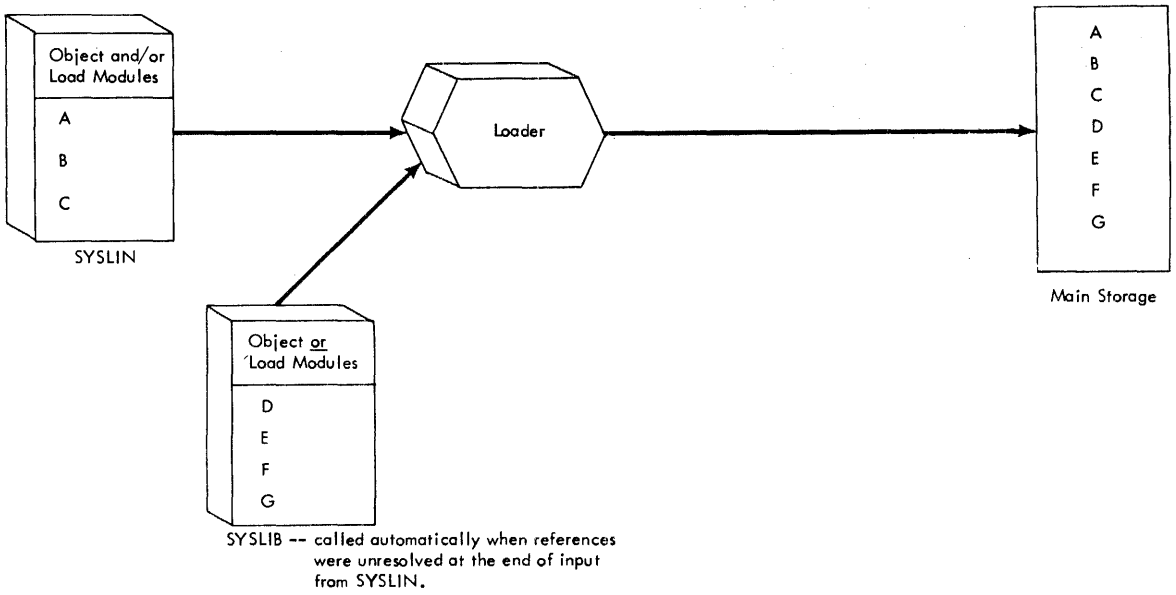


Figure 6-7. Loader Processing (SYSLIB Resolution)

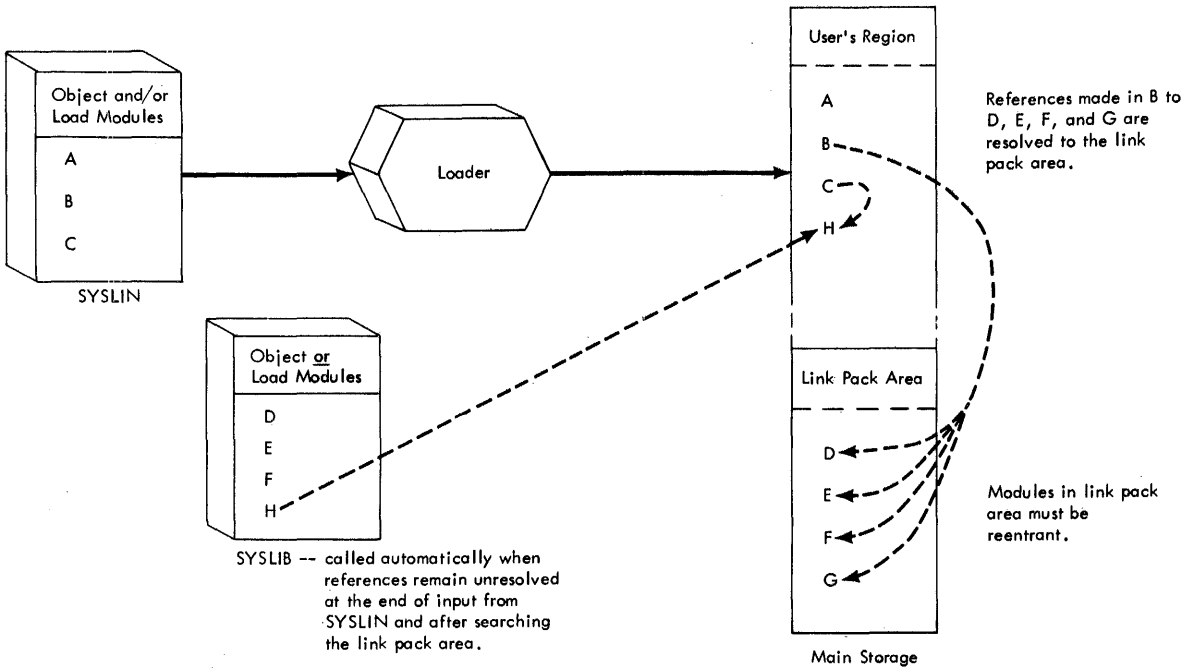


Figure 6-8. Loader Processing (Link-Pack Area and SYSLIB Resolution)

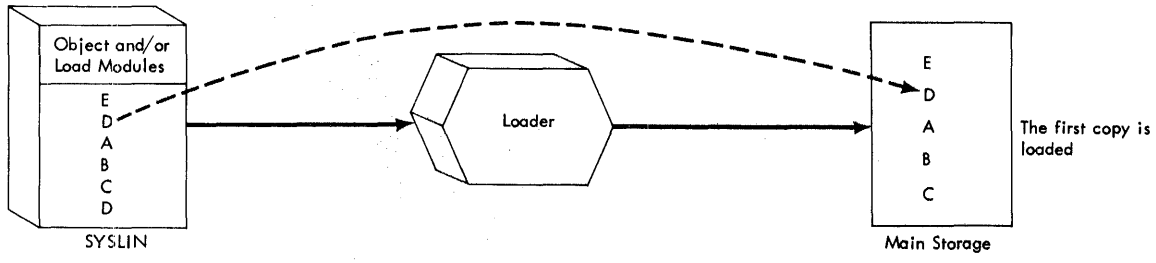


Figure 6-9. Automatic Editing

If you are using the MVT or MFT control program, the linkage loader will first search the link-pack area for library modules (see Figure 6-8) before searching the PL/I subroutine library. (The link-pack area is an area of main storage in which frequently used load modules are stored permanently; they can be accessed by any job running under MVT.) Library modules for PL1LIB must not be loaded into the link-pack area. If there is more than one copy of a program module in SYSLIN, the linkage loader will load the first one and ignore the rest (see Figure 6-9).

Main Storage Requirements

The minimum main storage requirements for the linkage loader are:

1. Storage for loader code: At least 10K bytes.

2. Storage for data management access method routines: At least 4K bytes.

3. Storage for buffers and tables used by the linkage loader: At least 3K bytes.

4. Storage for the program to be executed.

Thus the minimum main storage required when a program is to be processed by the linkage loader is at least 17K bytes for the linkage loader and its associated routines and data areas, and, in addition, whatever amount of main storage is required for the program to be executed. If the loader code and the data management access routines are stored in the link-pack area, then the amount of main storage required for program execution is 3K bytes for the loader data area and, in addition, the amount required for the problem program.

Job Control Language for Link-Loading

For most purposes, the IBM-supplied cataloged procedures are sufficent to provide the job control statements required for link-loading. However, you may want to supply your own job control statements or you may want to know which job control statements are required in order to construct your own cataloged procedures; therefore a brief discussion of the statements required or used by the linkage loader is given below.

The IBM-supplied cataloged procedures for the linkage loader are:

```
PL1LFG    Load-and-execute
PL1LFCG   Compile, load-and-execute
```

These are fully described in Chapter 8, 'Cataloged Procedures,' which also includes a description of the methods used to modify or override the statements in them.

EXEC STATEMENT

The name of the linkage loader program is IEWLDRGO; it also has the alias name LOADER. Either of these can be used in the basic EXEC statement:

```
// EXEC PGM=IEWLDRGO
// EXEC PGM=LOADER
```

The alias name LOADER is used in the IBM-supplied cataloged procedures; this name will be used for references to the linkage loader program in the remainder of this section.

By using the PARM parameter of the EXEC statement, you can select one or more of the optional facilities available with the linkage loader; these are described in 'Optional Facilities,' below. The use of the other parameters of the EXEC statement is as described in Chapter 7, 'Executing the Load Module.'

DD STATEMENTS

The linkage loader always requires one standard data set. You must define this data set in a DD statement with the standard name SYSLIN.

Three other standard data sets are optional and if you use them, you must define them in DD statements with the standard names SYSLOUT, SYSPRINT, and SYSLIB. The four standard data sets are summarized in Figure 6-10. The ddnames SYSLIN, SYSLIB and SYSLOUT for the loader data sets are those specified at system generation. Other ddnames for these data sets may have been specified at system generation for your installation; if they have, your job control statements must use these ddnames in place of those given above. The IBM-supplied cataloged procedures PL1LFCG and PL1LFG use the ddnames shown above; your system programmer will have to modify these procedures if the ddnames that apply at your installation are different.

Primary Input (SYSLIN)

Input to the linkage loader must be sequential and may be one of the following:

1. Object module:
   a. Output from the compiler.
   b. One member of a partitioned data set containing object modules.
2. Load Module:

   One member of a partitioned data set containing load modules.

ddname	Function	Possible Device Classes[1]
SYSLIN	Primary input data, normally the output from the compiler	SYSSQ or the input job stream (specified by DD *)
SYSLOUT	Loader messages and module map listing	SYSSQ, SYSDA, or SYSOUT=A
SYSPRINT	PL/I execution-time messages and problem program output listing	SYSSQ, SYSDA, or SYSOUT=A
SYSLIB	Automatic call library (usually the PL/I subroutine library)	SYSDA

[1]SYSSQ Magnetic-tape or direct-access device
SYSDA Direct-access device
SYSOUT=A Normal printed output class for system output

Figure 6-10. Linkage-Loader Data Sets

3. A concatenation of object modules, load modules or a mixture of both types.

The IBM-supplied cataloged procedure PL1LFCG includes a SYSLIN DD statement; if you want to modify this statement, you must refer to it by the qualified ddname GO.SYSLIN. The IBM-supplied cataloged procedure PL1LFLG does not include a SYSLIN DD statement; you must supply one, using the qualified ddname GO.SYSLIN.

Automatic Call Library (SYSLIB)

The SYSLIB data set is searched to resolve those external references that remain when all the external symbols that refer to locations within the program module have been resolved. This data set is always a partitioned data set; usually it is the PL/I subroutine library, SYS1.PL1LIB, but any library can be used provided it has the correct ddname. Libraries can be concatenated; the first library DD statement must have the ddname SYSLIB, the others must appear immediately after it in the job stream and must have the name field blank. The concatenated libraries can contain object or load modules but not a mixture of both.

The IBM-supplied cataloged procedures PL1LFCG and PL1LFG both include a SYSLIB DD statement. If you want to modify this statement, you must refer to it using the qualified ddname GO.SYSLIB.

Loader Listing (SYSLOUT)

The messages produced by the linkage loader, the module map that can be produced by the MAP option (see 'Optional Facilities,' below), and other information related to the linkage loader program can be written onto this data set. The SYSLOUT data set must be CONSECUTIVE organization and must have been specified at system generation; otherwise, all the loader information will be put out on SYSPRINT. A printed listing can be produced by allocating this data set to the system output class associated with a printer. This is usually designated by:

```
//SYSLOUT DD SYSOUT=A
```

The IBM-supplied cataloged procedures PL1LFCG and PL1LFG include this statement; if you want to modify the statement, you must refer to it with qualified ddname GO.SYSLOUT.

PL/I Execution-Time Messages and Problem Program Listing (SYSPRINT)

PL/I execution-time messages and output produced by the problem program are written on this data set. The data set must be of CONSECUTIVE organization. A printed listing can be obtained in the same way as for the loader information:

```
//SYSPRINT DD SYSOUT=A
```

The IBM-supplied cataloged procedures PL1LFCG and PL1LFG both include this statement; if you want to modify it, you must refer to it using the qualified ddname GO.SYSPRINT.

Examples

A typical sequence of job control statements for a compile-load-and-go job is:

```
//PGEX1 JOB
        .
        .
        .
// EXEC PGM=LOADER
//SYSLIN DD DSNAME=*.PL1L.SYSLIN,
//             DISP=(OLD,DELETE)
//SYSLIB DD DSNAME=SYS1.PL1LIB,DISP=SHR
//SYSLOUT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
```

Here a PL/I program has been compiled in a job step with the step name PL1L; the resultant object module has been placed in the SYSLIN data set. Because this module is to be loaded and executed in the same job as the compilation, the SYSLIN statement can use the backwards reference as shown. If the compilation and load-and-go steps were in different jobs, the SYSLIN reference would have to specify a permanent data set, cataloged or uncataloged.

The IBM-supplied cataloged procedure PL1LFCG includes SYSLIN statements in both the compile and the load-and-go steps; you do not need to specify this statement unless you want to modify it. The IBM-supplied cataloged procedure PL1LFG does not include a SYSLIN statement; you must supply one, using the qualified ddname GO.SYSLIN.

A more complicated example is given below; it has three concatenated input data sets (one of which is an object deck), and two libraries to be searched for external references. The job control statements differ slightly, according to the control program used:

PCP

```
// EXEC PGM=LOADER
//SYSLIN DD DSNAME=OBJMOD,
//             UNIT=SYSSQ,
//             VOLUME=SER=30103,
//             DISP=(OLD,KEEP)
// DD DSNAME=MODLIB(MOD55),DISP=SHR
// DD DDNAME=IN
//SYSLIB DD DSNAME=SYS1.PL1LIB,
//             DISP=SHR
// DD DSNAME=PRIVLIB,DISP=SHR
//SYSLOUT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//IN DD *

Object deck to be input to the linkage loader

/*
```

Here the input data sets are:

1. OBJMOD: An uncataloged data set containing the object module output from a compilation.

2. MODLIB: A cataloged data set containing a library of load modules; the member to be included here has the name MOD55.

3. IN: An object deck to be included in the job stream.

The two libraries to be searched for external references are the PL/I subroutine library (SYS1.PL1LIB) and a private library, PRIVLIB.

The IN DD statement shown here is only required if you want to include in the job stream an object deck for input to the linkage loader. The statement

```
// DD DDNAME=IN
```

specifies that the IN data set is to be concatenated with other data sets to form the SYSLIN data set. The statement

```
//IN DD *
```

specifies that the IN data set is to be read from the job stream. Because PCP does not permit the use of more than one data set in the input stream for a single job step, you therefore cannot assign the input stream data to more than one data set. This means that you cannot have execution data in the input stream when there is an object deck, concatenated as shown, in the input stream.

Note: If execution data in the input stream is required when using an object deck, the linkage editor must be used so that each input stream data set is in a separate job step.

If you use the cataloged procedures PL1LFCG and PL1LFG, you can modify or add to the statements in these procedures to generate the sequence described above. Modified or added statements must be referred to in the correct order; this is:

PL1LFCG:

```
PL1L.SYSLIN (if you want to modify the
             data set for the output from
             the compiler)
GO.SYSLIB
GO.SYSLIN
GO.SYSIN
GO.IN
```

PL1LFG:

```
  GO.SYSLIB
  GO.SYSLIN
  GO.SYSIN
  GO.IN
```

MFT or MVT

```
// EXEC PGM=LOADER
//SYSLIN DD DSNAME=OBJMOD,
//             UNIT=SYSSQ,
//             VOLUME=SER=30104,
//             DISP=(OLD,KEEP)
// DD DSNAME=MODLIB(MOD55),DISP=SHR
// DD DDNAME=IN
//SYSLIB DD DSNAME=SYS1.PL1LIB,
//             DISP=SHR
// DD DSNAME=PRIVLIB,DISP=SHR
//SYSLOUT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//IN DD *

Object deck for input to linkage loader

/*
//SYSIN DD *

Execution data, if any, to be placed here

/*
```

The input data sets and the two libraries are as described above.

If you want to include, in the job stream, both an object deck and execution data, the job control language is less complicated than with PCP because both MFT and MVT permit the use of more than one data set in the input stream for the same job step. Hence both IN and SYSIN can have the DD * notation.

If you use the cataloged procedures PL1LFCG or PL1LFG, you can modify or add to the statements in those procedures to generate the sequence described above. Modified or added statements must be referred to in the correct order; this is:

PL1LFCG:

```
  PL1L.SYSLIN (if you want to modify the
               data set for the output from
               the compiler)
  GO.SYSLIB
  GO.SYSLIN
  GO.IN          these can be in any order
  GO.SYSIN
```

PL1LFG:

```
  GO.SYSLIB
  GO.SYSLIN
  GO.IN          these can be in any order
  GO.SYSIN
```

Optional Facilities

CONTROL STATEMENTS

The linkage loader does not support the linkage editor control statements (ALIAS, CHANGE, ENTRY, INCLUDE, INSERT, LIBRARY, NAME, OVERLAY, REPLACE, SETSSI). The presence of any of these in the job stream will not be treated as an error; the job will continue to be processed, and the name of the statement is printed on SYSLOUT together with a diagnostic message.

The linkage loader compensates for the absence of the facilities provided by these control statements by allowing the concatenation of both object and load modules in SYSLIN, and by allowing an entry point to be specified in the EP option of the PARM parameter (see below).

OPTIONS IN THE PARM PARAMETER

The linkage loader provides a number of optional facilities that you can select by including the appropriate keywords from the following list in the PARM parameter of the EXEC statement that invokes it:

```
CALL|NOCALL|NCAL
EP
LET|NOLET
MAP|NOMAP
PRINT|NOPRINT
SIZE
RES|NORES
```

If any other keywords appear in the PARM parameter, they will be ignored and when processing is complete the keywords and a diagnostic message will be printed on SYSLOUT.

PARM Parameter Format

The basic format of the PARM parameter field is:

```
PARM='[option list][/pgmparm]'
```

where 'option list' is a list of linkage loader options, and
'pgmparm' is a parameter to be passed to the main procedure of the PL/I program to be executed.

The conventions for passing a parameter to the main procedure of a PL/I program are described in Chapter 7. In the examples

given below, the program parameter is referred to as PP.

If both loader options and a program parameter occur in the PARM parameter, the loader options are given first and are separated from the program parameter by a slash. If there are loader options but no program parameter, the slash is omitted. If there is more than one option, the option keywords are separated by commas.

The PARM parameter field can have one of three formats:

1. If the special characters '/' or '=' are used, the field must be enclosed in single quotes:

   ```
   PARM='MAP,EP=FIRST/PP'
   PARM='MAP,EP=FIRST'
   PARM='/PP'
   ```

2. If these characters are not included, and there is more than one loader option, the options must be enclosed in parentheses:

   ```
   PARM=(MAP,LET)
   ```

3. If these characters are not included, and there is only one loader option, neither quotes nor parentheses are required:

   ```
   PARM=MAP
   ```

If you want to modify the PARM parameter options specified in a cataloged procedure, you must refer to the PARM parameter by the qualified name PARM.procstepname, for example, PARM.GO.

CALL|NOCALL|NCAL

The CALL option specifies that an attempt to resolve external references will be made by an automatic search of the data set named in the SYSLIB DD statement. If this statement does not exist in the job stream, the option is ignored.

The NOCALL and NCAL options specify that no automatic search will be made. The term 'NCAL' is included to preserve compatibility with the linkage editor.

EP

The EP option specifies the entry-point name.

```
EP=epname
```

where 'epname' is an external name which is to be assigned as the entry point of the program to be executed. If all the input modules are load modules, you must specify EP=IHENTRY.

LET|NOLET

The LET option specifies that the linkage loader will try to execute the problem program even if a severity 2 error condition has been found.

The NOLET option specifies that, if a severity 2 error condition has been found, the linkage loader will not execute the problem program.

MAP|NOMAP

The MAP option specifies that the linkage loader will produce, on SYSLOUT, a map of the load module to be executed. This map includes a list of the external names and their absolute addresses, a list of the pseudo-registers, the total length of the module, and the absolute address of its entry point. If there is no SYSLOUT DD statement, the option will be ignored. The module map is described fully in 'Listing' below.

PRINT|NOPRINT

The PRINT option specifies that the SYSLOUT data set will be opened and used for diagnostic messages and other linkage loader information.

The NOPRINT option specifies that the SYSLOUT data set will not be opened.

SIZE

The SIZE option specifies the maximum amount of main storage, in bytes, that can be used by the linkage loader to process and execute the problem program. Code the option in one of the following formats:

```
SIZE=10K
SIZE=(10K)
SIZE=10240
SIZE=(10240)
```

RES|NORES

The RES option specifies that an attempt to resolve external references will be made by an automatic search of the link-pack area. This search will be made after the primary input has been processed but before the SYSLIB data set is opened.

The NORES option specifies that the link-pack area will not be searched.

DEFAULT OPTIONS

Defaults for all the options except EP can be established at system generation. If no such defaults were specified, the linkage loader assumes defaults as follows:

```
CALL, NOLET, NOMAP, PRINT, SIZE=100K,
RES
```

It is your responsibility to find out the defaults in operation at your installation.

Listing

The linkage loader can provide listings on the SYSLOUT data set; the SYSPRINT data set is used by the problem program. The contents of each is:

Data Set	Contents
SYSLOUT	Linkage loader explanatory messages and diagnostic messages, and (optionally) a module map
SYSPRINT	PL/I execution-time diagnostic messages; problem program output

The SYSLOUT listing is described here; the SYSPRINT listing is described in Chapter 7, 'Executing the Load Module.'

The items in the SYSLOUT listing appear in the following sequence:

1. A statement identifying the level of linkage loader used.
2. Module map (if specified).
3. Explanatory error or warning messages.
4. Diagnostic messages.

MODULE MAP

If the MAP option is specified, a module map appears in the SYSLOUT listing. The map lists all the control sections in the module to be executed, and all the entry names (other than the first one) in each control section. The information for each reference is:

1. The control-section or entry-point name.
2. An asterisk, if the control section is in a module loaded from the SYSLIB data set.
3. An identifier, as follows:

   SD Section definition: the name of the control section.

   LR Label reference: identifying an entry point in the control section other than the primary entry point.

   CIM Common area: an EXTERNAL file, or a non-string element variable declared STATIC EXTERNAL without the INITIAL attribute.

4. Absolute address of the control section or entry point.

Each reference is printed left to right across the page and starts at a tab position. This gives the impression that the references are arranged in columns, but the correct way to read the map is line-by-line, not down each column.

The module map is followed by a similar listing of the pseudo-registers. The identifier used here is PR, and the address if the offset from the beginning of the pseudo-register vector (PRV). The total length of the PRV is given at the end.

The total length of the module to be executed, and the absolute address of its primary entry point, are given after the explanatory messages and before the diagnostic messages.

EXPLANATORY ERROR OR WARNING MESSSAGES

The linkage loader always lists details of any error or warning conditions that it discovers during processing. The format of the messages is given in 'Control Statements and Errors,' in the linkage editor section of this chapter.

When the module to be executed has been processed, the linkage loader prints out in full all the diagnostic messages referred to above, IBM System/360 Operating System: Linkage ditor and Loader contains explanations of these messages and the probable cause of the errors noted in them, and suggests how to rectify these errors.

The warning message IEW1001 almost always appears in the listing. The explanation for this is the same as that for IEW0461, described above in 'Diagnostic Message Directory,' in the linkage editor section of this chapter.

Chapter 7: Executing the Load Module

Introduction

To execute a program, it must be in the form of a load module or an object module. If in load-module form you must use an EXEC statement to request the job scheduler to load and execute. If in object-module form you must use an EXEC statement to request the job scheduler to load and execute the loader which will process the module and pass control to it.

Modules for execution are selected from one of two sources:

1. A partitioned data set which is a module library. The modules are either object modules created by the compiler or load modules created by the linkage editor.

2. A sequential data set which is an object module created by the compiler.

Partitioned data sets and module libraries are described in Chapter 12, sequential data sets are described in Chapter 9.

This chapter describes the selection of the object or load module for execution, the job control statements required for load module execution, and the messages and other data printed on the output listing.

Load Module Processing

IDENTIFYING THE MODULE

The data set containing the module to be selected for execution is identified in one of two ways:

1. Jobs using the linkage editor: The data set is identified in the PGM parameter of the EXEC statement for the execution job step.

2. Jobs using the linkage loader: The data set is identified in a DD statement with the name SYSLIN in the execution job step.

Jobs Using the Linkage Editor

The data set exists in a library created in a previous job step of the same job, or in a previous job.

Library Created in a Previous Job Step: The basic reference is:

```
//stepname EXEC PGM=*.prevstepname.ddname
```

where 'prevstepname' is the name of the job step in which the library is created.

If the data set is the output data set created by the linkage editor, code:

```
//stepname EXEC PGM=*.linkname.SYSLMOD
```

where 'linkname' is the link-edit stepname.

If you use the cataloged procedures PL1LFCLG or PL1LFLG, the code generated is:

```
//GO EXEC PGM=*.LKED.SYSLMOD
```

Library Created in a Previous Job: If the library is a system library (SYS1.LINKLIB), code:

```
//stepname EXEC PGM=progname
```

where 'progname' is the member name of the module in the system library.

If the library is a private library used as a job library, the syntax is the same as for the system library. The private library must be identified in a JOBLIB DD statement placed immediately after the JOB statement.

If the library is a private library used as a step library, the syntax is the same as for the system library. The private library must be identified in a STEPLIB DD statement which follows the EXEC statement that initiates the job step.

Jobs Using the Linkage Loader

The data set exists in a library or is a single module, created in a previous job step of the same job or in a previous job.

Library Created in a Previous Job Step: The basic reference is:

```
//SYSLIN DD DSNAME=*.stepname.ddname,
//             DISP=(disp)
```

where '*.stepname.ddname' refers to the name of the DD statement that describes the data set in which the member of the library is created.

'disp' is the set of terms for the disposition of the data set before and after the job step.

If you use the cataloged procedure PL1LFCG with input from a library, you must override the SYSLIN DD statements in both job steps. The ddnames for the input data set you are using must be qualified as PL1L.SYSLIN and GO.SYSLIN respectively. If you use the cataloged procedure PL1LFG, you must supply the SYSLIN DD statement, using //GO.SYSLIN.

Library Created in a Previous Job: If the library is the system library, code:

```
//SYSLIN DD DSNAME=dsname,DISP=(disp)
```

where 'dsname' is the name of the member of the library.

'disp' is as defined previously in this chapter.

If the library is a private library, the syntax is the same as for the system library.

If the library is neither cataloged nor in a job library, it must be described fully in the SYSLIN DD statement. (A cataloged data set, which can be a library or a single module, has its name in the system catalog and can be called by specifying the name only.) Code:

```
//SYSLIN DD DSNAME=dsname(mbname),
//             DISP=(disp),
//             VOLUME=volume,
//             UNIT=unit
```

where 'dsname' is the name of the library.

'mbname' and 'disp' are as defined previously in this chapter.

'volume' is the set of terms defining the volume and its usage.

'unit' specifies the storage device.

If you use the cataloged procedure PL1LFCG or PL1LFG, the same considerations apply as for a library created in a previous job step.

Module Created in a Previous Job Step: The basic reference is:

```
//SYSIN DD DSNAME=dsname,DISP=(disp)
```

where 'dsname' is the name of the temporary or permanent data set.

'disp' is as defined previously in this chapter.

The cataloged procedure PL1LFCG generates the appropriate SYSLIN DD statement. If you use the cataloged procedure PL1LFLG, you must supply this statement, using the qualified ddname //GO.SYSLIN.

Module Created in a Previous Job: The reference for the SYSLIN DD statement depends on whether the data set is cataloged or not. For a cataloged data set, code:

```
//SYSLIN DD DSNAME=dsname,DISP=(disp)
```

where 'dsname' is the name of the cataloged data set.

'disp' is as defined previously in this chapter.

If the data set is uncataloged, code:

```
//SYSLIN DD DSNAME=dsname,
//             DISP=(disp),
//             VOLUME=volume,
//             UNIT=unit
```

where 'dsname' is the name of the data set.

'disp', 'volume', and 'unit' are as defined previously in this chapter.

If you use the cataloged procedures PL1LFCG or PL1LFG, the same considerations apply as for libraries created in a previous job step.

Job Control Language for Execution

You can use a cataloged procedure to generate the job control statements or you can supply them yourself. The IBM-supplied cataloged procedures that apply to this step are:

PL1LFCG	Compile, load-and-execute
PL1LFCLG	Compile, link-edit, and execute
PL1LFG	Load-and-execute
PL1LFLG	Link-edit, and execute

These procedures are described in Chapter 8, 'Cataloged Procedures,' which includes an account of the methods used to modify or overwrite any of the statements in the procedures.

If you want to supply your own job control statements, the statements required for the execution job step are described below.

EXEC STATEMENT

The basic form of the EXEC statement requires only the PGM parameter:

```
// EXEC PGM=reference
```

where 'reference' has one of two forms:

1. Jobs using the linkage editor: a reference to the data set containing the load module. This has already been described in 'Identifying the Load Module' in this chapter.

2. Jobs using the linkage loader: the name of the loader program:

   ```
   PGM=LOADER
   ```

   The use of the linkage loader is described in Chapter 6, 'Linkage Editor and Loader.'

While the PGM parameter is the only mandatory parameter as far as the operating system is concerned, some or all of the other parameters available may be mandatory at your installation. The use of these parameters is discussed here.

ACCT Parameter

The accounting procedure at your installation may require you to provide information here, if each job step is to be charged separately.

COND Parameter

This is a useful parameter if your execution job step is dependent on the successful completion of a previous job step. The use of the EVEN and ONLY subparameters allows you precise control over the conditions under which this job step can be executed.

PARM Parameter

The PL/I (F) compiler provides a facility for passing, in this job step, a single parameter to the main procedure of the PL/I program. The data specified in the PARM parameter field can be up to 100 characters long and must be enclosed in quotation marks. Any character in the character set available can be specified. The associated parameter in the main procedure should be declared CHARACTER(100) VARYING.

The PARM parameter is a useful means of passing data to a load module. For example, if the SYSIN file is already associated with a user data set, any extra data required can be quickly inserted by being specified in this parameter. Another use is to pass data that can be employed to determine how the program will be executed and which category or categories of output will be produced. Both the PL/I (F) compiler and the linkage editor use the PARM parameter in this way; the characters passed represent various options which determine, for example, the information to be printed on the output listing.

If you want to use the PARM parameter for the second of these purposes, you should include, at the beginning of your PL/I program, code to convert the parameter characters to variables, and code to set a series of program switches. For example:

```
STOCK: PROC(INPARM) OPTIONS(MAIN);
       DCL INPARM CHAR(100) VAR,
           1 VALUES BASED(P),
                2 (TERMA,
                  TERMB,
                  TERMC,
                  TERMD,
                  TERME) CHAR(1),
                2 NUMBER PIC'9999',
                2 SPEC CHAR(1),
                2 CTCHAR CHAR(3),
                .
                .
                .
       P=ADDR(INPARM);
L1:IF  TERMA=TERMB THEN GO TO...;
       ELSE IF CTCHAR='I/M' THEN...;
L2:IF  TERMC1=TERMD THEN GO TO...;
       ELSE GO TO...;
L3:IF  TERMA=TERME THEN GO TO...;
       ELSE IF SPEC='*' THEN...;
L4:IF  NUMBER≥1000 THEN...;
       ELSE IF NUMBER<10 THEN...;
       .
       .
       .
END STOCK;
```

If this program, once compiled and link-edited (or loaded), is executed with the following EXEC statement:

```
// EXEC PGM=...,PARM='ABCDE1414*I/M'
```

then the elements of the structure variable VALUES will have the following values:

```
TERMA   =  'A'
TERMB   =  'B'
TERMC   =  'C'
TERMD   =  'D'
TERME   =  'E'
NUMBER  =  '1414'
SPEC    =  '*'
CTCHAR  =  'I/M'
```

These values will then be used in the four switching statements to determine which parts of the program will be executed and therefore what output the program will produce.

The result of the pointer assignment statement is that the elements in VALUES now have the appropriate values from the parameter field. If any character in the parameter field is omitted for a particular job, it must be replaced by a blank; this will create a blank in the corresponding VALUES element. The existence of a blank or blanks in these variables must be considered when the switching statements are coded, in order to avoid erroneous switching.

GET STRING or SUBSTR could be used in place of the pointer assignment, if the context was such that they offered an advantage.

DPRTY Parameter

If your load module is executed using the MVT control program, you can specify the priority for each job step. For details of this usage, see 'MVT Control Program' in Chapter 4, 'Job Initialization.'

The priority value specified in the EXEC statement is always overridden by a priority value specified in the JOB statement; if no priority is specified in the JOB statement, the value specified in the EXEC statement is assumed. If neither statement includes a priority parameter, the installation default (if any) is applied.

REGION Parameter

If your load module is executed using the MVT control program, you can specify a region size for the job step. For details of this usage, see 'MVT Control Program' in Chapter 4, 'Job Initialization.'

A region size specified in the EXEC statement is always overriden by a region size specified in the JOB statement. If a region size is not specified in the JOB statement, the size specified in the EXEC statement is assumed. If neither is specified, the installation default (if any) is applied.

ROLL Parameter

If your load module is executed using the MVT control program, you can obtain extra space dynamically in main storage by means of this parameter.

TIME Parameter

If your load module is executed using the MVT control program, you can specify the maximum time that this job step can use the CPU, by means of this parameter. The time specified here overrides the default time for the job class. This parameter is useful if there is a possibility that your program might go into a permanent loop or if your program requires a long execution time.

STANDARD DD STATEMENTS

Three standard data sets can be used for the execution job step. These are:

Purpose	ddname
Input	SYSIN
Output	SYSPRINT
Dump	SYSABEND, SYSUDUMP, or PL1DUMP

Of these, only SYSPRINT is necessary in every job. If it is omitted, the system messages will be put out on the operator's console and the other output data will be lost. If your installation has multiple-console support (MCS), you must find out on which console or consoles the system messages will appear.

Input (SYSIN)

If you want to include data in the input stream, you can do so by means of a DD statement of the form: //ddname DD * immediately preceding the data. (A data set in the input stream does not have to be called SYSIN; it can have any name.) The data should be 80-byte F-format unblocked records (for example, punched cards). Code:

```
//SYSIN DD *
input data
/*
```

The SYSIN DD statements with this notation must be the last statement in the job control statements for the job step for PCP. If the data includes the characters // in columns 1 and 2 replace //SYSIN DD * by //SYSIN DD DATA.

The IBM-supplied cataloged procedures PL1LFCG, PL1LFCLG, PL1LFG, and PL1LFLG do not include a SYSIN DD statement. If you want to use one, you must qualify the ddname with the step name, that is, code //GO.SYSIN.

Output (SYSPRINT)

System and problem program output can be put out through the system output streams. The advantage of this is that each output stream can be associated with an output device, such as a printer; this device is specified in the SYSOUT parameter of the DD statement for the output data set. If, as is usual, output stream A is associated with a printer, the statement

```
//SYSPRINT DD SYSOUT=A
```

will result in all SYSPRINT data appearing in a printed listing.

The SYSPRINT data can include both system output (for example, diagnostic messages from job control statements) and problem program output. If you want system and problem program ouput to be on separate listings, you can arrange this by means of the MSGCLASS parameter in the JOB statement, see Chapter 4, 'Job Initialization.'

The IBM-supplied cataloged procedures PL1LFCG, PL1LFCLG, PL1LFG, and PL1LFLG include this statement in the execution job step.

Dump (SYSABEND, SYSUDUMP, or PL1DUMP)

If you want to obtain a printed listing of any of these dumps, code one of the following:

```
//SYSABEND DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//PL1DUMP DD SYSOUT=A
```

For further information on DD statements for dumps, see Chapter 14, 'Other Facilities of the Operating System'.

USER DD STATEMENTS

In execution of your program, you can create data sets or you can access data sets already created. For the types of data set that can be used with a PL/I program, see Chapter 9, 'Data Sets and PL/I Files.' For the methods of creating or accessing these data sets, see Chapter 10, 'Stream-Oriented Transmission,' and Chapter 11, 'Record-Oriented Transmission.'

The IBM-supplied cataloged procedures PL1LFCG, PL1LFCLG, PL1LFG, and PL1LFLG do not include statements for user data sets. If you want to include such statements, you must code them with the ddname qualified with the step name, that is, code //GO.ddname.

Listing

CONTENTS OF SYSPRINT LISTING

The SYSPRINT listing that you obtain with your job consists of some or all of the following entries, usually in the sequence given:

1. Job output, in the format established by any PUT FILE(SYSPRINT) statements in your PL/I source program.
2. Execution-time diagnostic messages produced by the PL/I library. These have the format:

   IHEdddI Message text

   where 'ddd' is a decimal number.
3. Dump of part of or all main storage. The important information here for you is the completion codes.

4. Diagnostic messages from any operating system facility which has an interface with your PL/I program. For example, checkpoint or sort messages would be printed here.

5. Job scheduler messages, showing the final disposition of the data sets used in the job. These messages are printed in pairs, as follows:

```
IEFdddI  Data set name  Disposition
IEFdddI  Volume serial number of
         data set
```

   where 'ddd' is a decimal number. The two messages numbers are always the same for each pair.

6. Step completion information. This depends on your installation but will probably contain the step name, step time, clock time, date and return code. The return code is always zero unless you include statements in your PL/I program to create a return code.

7. Job completion information (if this is the last step). This depends on your installation but will probably include job name, job time, clock time, and date.

Of these, only the job scheduler messages and the step and job completion information will appear in every job. The other items will depend on the nature of the source program and how successfully it executes.

RETURN CODES

The COND parameter causes the job scheduler to test the return codes put out at the end of every job step. These codes indicate the degree of success of the job step; they show, for example, whether the program can be expected to complete normally and whether the output data will be as required. Such codes are returned as a matter of course at the end of the compilation and linkage-editor job steps (see Chapters 5 and 6 respectively), therefore you do not have to take any action to supply them.

When the execution job step terminates, a return code is produced to indicate whether or not it terminated successfully.

Code	Termination
Zero	Normal
Non-zero (value depends on cause)	Abnormal

The return code is provided by the PL/I library error-handling subroutines. You can generate a return code in your program by coding the statements shown below; this code is added to the value provided by the PL/I library and is printed on the listing.

Single-task processing:

```
DCL IHESARC ENTRY(FIXED BINARY(31,0));
   .
   .
   .
CALL IHESARC(expression);
```

Multitask processing:

```
DCL IHETSAC ENTRY(FIXED BINARY(31,0));
   .
   .
   .
CALL IHETSAC(expression);
```

Note: The entry point IHETSAC is applicable to the major task only.

On evaluation, the expression supplies the required value for the return code. IHESARC and IHETSAC must be declared as fullword binary, otherwise errors may occur in the returned code because of the halfword binary feature.

Communication with Program during Execution

The DISPLAY statement provides a means of communicating with the load module for the PL/I program during execution. The program can put out, on the operator's console, a message of up to 72 characters in length. If a reply is expected, the syntax of the reply message must use a 2-digit code provided by the operating system. The operator uses this code as a prefix to the reply message.

If your installation has multiple-console support (MCS), you must find out on which console your messages will appear. The PL/I implementation of the route and descriptor codes in MCS is restricted to:

PL/I Statement	Route Code	Descriptor Code
DISPLAY	2	7
DISPLAY with REPLY	1	7

If SYSPRINT is not available, the console on which error messages will appear is designated by route code 11 and descriptor code 7. These error messages will be truncated to 72 characters if they exceed this length.

Chapter 8: Cataloged Procedures

Introduction

A cataloged procedure is a set of job control statements stored in a system library, the procedure library (SYS1.PROCLIB). It comprises one or more EXEC statements, each of which may be followed by one or more DD statements. You can retrieve the statements by naming the cataloged procedure in the PROC parameter of an EXEC statement in the input job stream. When the job scheduler encounters such an EXEC statement, it replaces it in the input stream with the statements of the cataloged procedure.

The use of cataloged procedures saves time and obviates errors in coding frequently used sets of job control statements. Even if the statements in a cataloged procedure do not match your requirements exactly, you can easily modify them or add new statements for the duration of a job.

This chapter describes seven cataloged procedures supplied by IBM for use with the (F) compiler, and explains how to invoke them and how to make modifications to them.

PL/I Cataloged Procedures supplied by IBM

The following paragraphs do not fully describe the individual statements of the IBM cataloged procedures, since all the parameters are discussed elsewhere in this manual. Note that the cataloged procedures described here are the standard PL/I cataloged procedures. It is recommended that each installation review these procedures and modify them to obtain the most efficient use of the facilities available and to allow for installation conventions; refer to 'Permanent Modification,' at the end of this chapter.

The standard PL/I cataloged procedures are:

PL1DFC	Compile and punch object deck
PL1LFC	Compile and place object module on magnetic-tape or direct-access device
PL1LFCL	Compile and link-edit
PL1LFCLG	Compile, link-edit, and execute
PL1LFLG	Link-edit and execute
PL1LFCG	Compile, load-and-execute
PL1LFG	Load-and-execute

COMPILE AND PUNCH OBJECT DECK (PL1DFC)

The cataloged procedure PL1DFC (Figure 8-1) comprises only one job step, in which the (F) compiler is executed with the DECK option. (IEMAA is the symbolic name of the compiler.) In common with the other cataloged procedures that include a compilation job step, PL1DFC does not include a DD statement for the compiler input data set; you must always supply an appropriate statement with the qualified ddname PL1D.SYSIN. Because the EXEC statement includes the options DECK and NOLOAD, the compiler will place the object module in card-image form in the data set defined by the DD statement SYSPUNCH; conventionally, the system output device of class B is always a card punch.

COMPILE AND WRITE OBJECT MODULE (PL1LFC)

The cataloged procedure PL1LFC (Figure 8-2) is similar to PL1DFC; it differs only in that the options specified for the compilation are LOAD and NODECK, and the DD

```
//PL1D EXEC PGM=IEMAA,PARM='DECK,NOLOAD',REGION=52K
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD SYSOUT=B
//SYSUT3 DD DSNAME=&&SYSUT3,UNIT=SYSDA,SPACE=(80,(250,250)),
//             DCB=BLKSIZE=80
//SYSUT1 DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(60,60),,CONTIG),
//             SEP=(SYSUT3,SYSPUNCH),DCB=BLKSIZE=1024
```

Figure 8-1. Cataloged Procedure PL1DFC (Compile and Punch Object Deck)

```
//PL1L EXEC PGM=IEMAA,PARM='LOAD,NODECK',REGION=52K
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSNAME=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//             SPACE=(80,(250,100))
//SYSUT3 DD DSNAME=&&SYSUT3,UNIT=SYSDA,SPACE=(80,(250,250)),
//             DCB=BLKSIZE=80
//SYSUT1 DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(60,60),,CONTIG),
//             SEP=(SYSUT3,SYSLIN),DCB=BLKSIZE=1024
```

Figure 8-2. Cataloged Procedure PL1LFC (Compile and Write Object Module)

```
//PL1L EXEC PGM=IEMAA,PARM='LOAD,NODECK',REGION=52K
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSNAME=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//             SPACE=(80,(250,100))
//SYSUT3 DD DSNAME=&&SYSUT3,UNIT=SYSDA,SPACE=(80,(250,250)),
//             DCB=BLKSIZE=80
//SYSUT1 DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(60,60),,CONTIG),
//             SEP=(SYSUT3,SYSLIN),DCB=BLKSIZE=1024
//LKED EXEC PGM=IEWL,PARM='XREF,LIST',COND=(9,LT,PL1L),REGION=96K
//SYSLIB DD DSNAME=SYS1.PL1LIB,DISP=SHR
//SYSLMOD DD DSNAME=&&GOSET(GO),DISP=(MOD,PASS),UNIT=SYSDA,
//             SPACE=(1024,(50,20,1),RLSE)
//SYSUT1 DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(200,20)),
//             SEP=(SYSLMOD,SYSLIB),DCB=BLKSIZE=1024
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//       DD DDNAME=SYSIN
```

Figure 8-3. Cataloged Procedure PL1LFCL (Compile and Link-Edit)

statement SYSLIN replaces SYSPUNCH. The LOAD option causes the compiler to place the object module, in a form suitable for input to the linkage editor, in the data set defined by the DD statement SYSLIN. This DD statement defines a temporary data set named &&LOADSET, since the cataloged procedure assumes that a link-edit job step will follow. If you want to retain the load module, you must substitute your own DD statement for the one supplied. Input data for this cataloged procedure requires the qualified ddname PL1L.SYSIN.

COMPILE AND LINK-EDIT (PL1LFCL)

The cataloged procedure PL1LFCL (Figure 8-3) comprises two job steps: PL1L, which is identical with cataloged procedure PL1LFC, and LKED, which invokes the linkage editor (symbolic name IEWL) to link-edit the object module produced in the first step.

Input data for the compilation job step requires the qualified ddname PL1L.SYSIN.

The COND parameter in the EXEC statement LKED specifies that this job step should be bypassed if the return code produced by the compiler is greater than 9 (that is, if a severe or termination error occurred during compilation).

The DD statement SYSLIB specifies the PL/I subroutine library, from which the linkage editor will read appropriate modules for inclusion in the load module.

The linkage editor always places the load modules it creates in the library defined by the DD statement SYSLMOD. This statement in the cataloged procedure specifies a new temporary library &&GOSET, in which the load module will be placed and given the member name GO (unless you specify the OBJNM option for the compilation). In specifying a temporary library, the cataloged procedure assumes that you will execute the load module in the same job; if you want to retain the module, you must substitute your own statement for the DD statement SYSLMOD.

The statement DDNAME=SYSIN following the DD statement SYSLIN allows you to concatenate a data set defined by a DD statement with the name SYSIN with the primary input to the linkage editor; for example, you could place linkage-editor control statements in the input stream by this means.

```
//PL1L EXEC PGM=IEMAA,PARM='LOAD,NODECK',REGION=52K
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSNAME=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//             SPACE=(80,(250,100))
//SYSUT3 DD DSNAME=&&SYSUT3,UNIT=SYSDA,SPACE=(80,(250,250)),
//             DCB=BLKSIZE=80
//SYSUT1 DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(60,60),,CONTIG),
//             SEP=(SYSUT3,SYSLIN),DCB=BLKSIZE=1024
//LKED EXEC PGM=IEWL,PARM='XREF,LIST',COND=(9,LT,PL1L),REGION=96K
//SYSLIB DD DSNAME=SYS1.PL1LIB,DISP=SHR
//SYSLMOD DD DSNAME=&&GOSET(GO),DISP=(MOD,PASS),UNIT=SYSDA,
//             SPACE=(1024,(50,20,1),RLSE)
//SYSUT1 DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(200,20)),
//             SEP=(SYSLMOD,SYSLIB),DCB=BLKSIZE=1024
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//       DD DDNAME=SYSIN
//GO EXEC PGM=*.LKED.SYSLMOD,COND=((9,LT,LKED),(9,LT,PL1L))
//SYSPRINT DD SYSOUT=A
```

Figure 8-4. Cataloged Procedure PL1LFCLG (Compile, Link-Edit, and Execute)

COMPILE, LINK-EDIT, AND EXECUTE (PL1LFCLG)

The cataloged procedure PL1LFCLG (Figure 8-4) comprises three job steps, PL1L and LKED, which are identical with the two job steps of PL1LFCL, and GO, in which the load module created in the step LKED is executed. The third step will be executed only if no severe or termination errors occur in the preceding steps.

Input data for the compilation job step requires the qualified ddname PL1L.SYSIN; input data for the execution job step requires the name GO.SYSIN.

LINK-EDIT AND EXECUTE (PL1LFLG)

The cataloged procedure PL1LFLG (Figure 8-5) comprises two job steps, LKED and GO, which are similar to the steps of the same names in PL1LFCLG. In the job step LKED, the DD statement SYSLIN does not define a data set, but merely refers the job scheduler to the DD statement SYSIN, which you must supply with the qualified ddname LKED.SYSIN. This DD statement defines the data set from which the linkage editor will obtain its primary input. Execution of the step GO is conditional on successful execution of the step LKED only.

COMPILE, LOAD AND EXECUTE (PL1LFCG)

The cataloged procedure PL1LFCG (Figure 8-6) achieves the same result as PL1LFCLG. However, instead of using three job steps (compile, link-edit, and execute), it has only two (compile, and load-and-execute). In the second job step of PL1LFCG, the operation system loader program is executed; this program link-edits the object program produced by the compiler and then executes the load module immediately.

Input data for the compilation job step requires the qualified ddname PL1L.SYSIN.

Note that the REGION parameter of the EXEC statement GO specifies 96K bytes. Since the loader requires about 17K bytes of main storage, there are about 79K bytes for your program; if this is likely to be insufficient, you must modify the REGION parameter.

```
//LKED EXEC PGM=IEWL,PARM='XREF,LIST',REGION=96K
//SYSLIB DD DSNAME=SYS1.PL1LIB,DISP=SHR
//SYSLMOD DD DSNAME=&&GOSET(GO),DISP=(MOD,PASS),UNIT=SYSDA,
//             SPACE=(1024,(50,20,1),RLSE)
//SYSUT1 DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(200,20)),
//             SEP=(SYSLMOD,SYSLIB),DCB=BLKSIZE=1024
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DDNAME=SYSIN
//GO EXEC PGM=*.LKED.SYSLMOD,COND=(9,LT,LKED)
//SYSPRINT DD SYSOUT=A
```

Figure 8-5. Cataloged Procedure PL1LFLG (Link-Edit and Execute)

```
//PL1L EXEC PGM=IEMAA,PARM='LOAD,NODECK',REGION=52K
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSNAME=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//             SPACE=(80,(250,100))
//SYSUT3 DD DSNAME=&&SYSUT3,UNIT=SYSDA,SPACE=(80,(250,250)),
//             DCB=BLKSIZE=80
//SYSUT1 DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(60,60),,CONTIG),
//             SEP=(SYSUT3,SYSLIN),DCB=BLKSIZE=1024
//GO EXEC PGM=LOADER,PARM='MAP,PRINT',REGION=96K,COND=(9,LT,PL1L)
//SYSLIB DD DSNAME=SYS1.PL1LIB,DISP=SHR
//SYSLIN DD DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//SYSLOUT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
```

Figure 8-6. Cataloged Procedure PL1LFCG (Compile, Load-and-Execute)

```
//GO EXEC PGM=LOADER,PARM='MAP,PRINT',REGION=96K
//SYSLIB DD DSNAME=SYS1.PL1LIB,DISP=SHR
//SYSLOUT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
```

Figure 8-7. Cataloged Procedure PL1LFG (Load-and-Execute)

The use of the loader imposes certain restrictions on your PL/I program; before using this cataloged procedure, refer to Chapter 6, which explains how to use the loader.

LOAD AND EXECUTE (PL1LFG)

The cataloged procedure PL1LFG (Figure 8-7) achieves the same result as PL1LFLG. However, instead of using two job steps (link-edit and execute), it has only one. In this job step, the operating system loader program is executed. This program link-edits and executes an object program placed in a data set defined by a DD statement with the name SYSLIN; you must supply this DD statement with the qualified ddname GO.SYSLIN.

Note that the REGION parameter of the EXEC statement specifies 96K bytes. Since the loader requires about 17K bytes of main storage, there are about 79K bytes for your program; if this is not likely to be sufficient, you must modify the parameter.

The use of the loader imposes certain restrictions on your PL/I program; before using this cataloged procedure, refer to Chapter 6, which explains how to use the loader.

DEDICATED WORKFILES

All the cataloged procedures that require the use of workspace data sets in any job step include DD statements that permit the substitution of dedicated workfiles for these data sets. In such a DD statement, the dsname is coded:

```
DSNAME=&&ddname
```

where 'ddname' is the ddname of the dedicated workfile that is to be substituted for the workspace data set. This substitution only occurs if:

1. The job is being processed under MVT.
2. An initiator that generates dedicated workfiles is used to select the job.

Dedicated workfiles may be used for the SYSUT1 and SYSUT3 work data sets for the PL/I (F) compiler. Dedicated workfiles may also be used for the SYSLIN data set in compile-link-edit-and-go or compile-load-and-go jobs. However, care should be taken to specify a block size (which must be a multiple of 80 bytes) for this data set, as the compiler will not override an unsuitable block size left in the dedicated workfile DSCB by a previous job (this is because the compiler cannot know if SYSLIN is a permanent data set, perhaps erroneously specified, for which the block size may not be overriden). This block size may be specified either in user JCL or the cataloged procedure used. If it is specified in a cataloged procedure, care must be taken to ensure that the block size is overridden by user JCL when the procedure is used to update a permanent SYSLIN data set with a block size different to that specified in the procedure.

Note that the standard cataloged procedures PL1LFC, PL1LFCL, PL1LFCLG, and

PL1LFCG use the dsname LOADSET and specify UNIT=SYSSQ for SYSLIN. They assume that SYSLIN will not use a dedicated workfile. These may be modified to do so subject to the above restrictions on block size, and provided UNIT is changed to specify UNIT=SYSDA.

For a complete description of the use of dedicated workfiles, see IBM System/360 Operating System: System Programmer's Guide.

Using Cataloged Procedures

To invoke a cataloged procedure, specify its name in the PROC parameter of an EXEC statement. For example, to use the cataloged procedure PL1DFC, you could include the following statement in the appropriate position among your other job control statements in the input stream:

```
// EXEC PROC=PL1DFC
```

Note that you need not code the keyword PROC; if the first operand in the EXEC statement does not begin PGM= or PROC=, the job scheduler interprets it as the name of a cataloged procedure. Thus, the following statement is equivalent to that given above:

```
// EXEC PL1DFC
```

When the job scheduler encounters the name of a cataloged procedure in an EXEC statement, it extracts the statements of the cataloged procedure from the procedure library and substitutes them for the EXEC statement in the input job stream. If you include the parameter MSGLEVEL=1 in your JOB statement, the job scheduler will include the original EXEC statement in its listing, and will append the statements of the cataloged procedure. In the listing, cataloged procedure statements are identified by XX or X/ as the first two characters; X/ signifies a statement that has been modified for this invocation of the cataloged procedure.

An EXEC statement identifies a job step, which can require either the execution of a program or the invocation of a cataloged procedure. A cataloged procedure includes one or more EXEC statements, which identify procedure steps. However, an EXEC statement in a cataloged procedure cannot invoke another cataloged procedure; it must request the execution of a program. Thus a job comprises one or more job steps, each of which involve one or more procedure steps.

It will usually be necessary for you to modify the statements of a cataloged procedure for the duration of the job step in which it is invoked, either by adding DD statements to it or by overriding one or more parameters in the EXEC or DD statements. For example, all the cataloged procedures that involve compilation require the addition of a DD statement with the name SYSIN to define the data set that contains the source statements. Also, whenever you use more than one standard linkage-editor procedure step in a job, you must modify all but the first cataloged procedure that you invoke if you want to execute more than one of the load modules; this special case is discussed in 'Altering Cataloged Procedures,' below.

Altering Cataloged Procedures

You can modify a cataloged procedure permanently by rewriting the job control statements that are stored in the procedure library. Alternatively, you can make temporary changes by including parameters in the EXEC statement that invokes the cataloged procedure or by placing additional DD statements after the EXEC statement.

Permanent alterations should be made only by system programmers responsible for maintaining the procedure library. Some of the considerations that may influence their decisions as to whether and how to modify the standard cataloged procedures are discussed below.

Most programmers find it necessary to make temporary modifications whenever they use a cataloged procedure. Such changes apply only for the duration of the job step in which the procedure is invoked and only to that job step; they do not affect the copy of the cataloged procedure stored in the procedure library.

TEMPORARY MODIFICATION

Temporary modifications can apply to EXEC or DD statements in a cataloged procedure. To change a parameter of an EXEC statement, you must include a corresponding parameter in the EXEC statement that invokes the cataloged procedure; to change one or more parameters of a DD statement, you must include a corresponding DD statement after the EXEC statement that invokes the cataloged procedure. Although you may not add a new EXEC statement to a cataloged procedure, you can always include additional DD statements.

EXEC Statement

If a parameter of an EXEC statement that invokes a cataloged procedure has an unqualified name, that parameter applies to all the EXEC statements in the cataloged procedure. The effect on the cataloged procedure depends on the parameter:

PARM applies to the first procedure step and nullifies any other PARM parameters.

COND and ACCT apply to all the procedure steps.

TIME and REGION apply to all the procedure steps and override existing values.

For example, the statement

```
// EXEC PL1LFCLG,PARM='SIZE=999999',
//             REGION=144K
```

invokes the cataloged procedure PL1LFCLG, substitutes the option SIZE=999999 for LOAD and NODECK in the EXEC statement PL1L, and nullifies the PARM parameter in the EXEC statement LKED; it also specifies a regional size of 144K for all three procedure steps.

To change the value of a parameter in only one EXEC statement of a cataloged procedure, or to add a new parameter to one EXEC statement, you must identify the EXEC statement by adding its name as a suffix to the parameter name; separate the parameter name and the step name with a period. For example, to alter the region size for procedure step PL1L only in the preceding example, code:

```
// EXEC PROC=PL1LFCLG,
//             PARM='SIZE=999999',
//             REGION.PL1L=144K
```

A new parameter specified in the invoking EXEC statement overrides completely the corresponding parameter in the procedure EXEC statement. This is particularly important with the PARM parameter. For example, the statement

```
// EXEC PL1DFC,PARM.PL1L='SIZE=999999,
//             EXTREF'
```

would be in error if the standard compiler default options applied. The default options NODECK and LOAD would apply, rather than the options DECK and NOLOAD specified in the procedure EXEC statement. Consequently, the compiler would attempt to open the data set defined by the DD statement SYSLIN, which does not exist in the cataloged procedure PL1DFC.

You can suppress all the options specified by a parameter by coding the keyword and equal sign without a value. For example, to suppress the bulk of the linkage-editor listing when invoking the cataloged procedure PL1LFCLG, code:

```
// EXEC PL1LFCLG,PARM.LKED=
```

DD Statement

To add a new DD statement to a cataloged procedure, or to modify one or more parameters of an existing DD statement, you must include, in the appropriate position in the input stream, a DD statement with a name of the form 'procstepname.ddname'. If 'ddname' is the name of a DD statement already present in the procedure step identified by 'procstepname,' the parameters in the new DD statement override the corresponding parameters in the existing DD statement; otherwise, the new DD statement is added to the procedure step. For example, the statement

```
//PL1D.SYSIN DD *
```

adds a DD statement to the step PL1D of cataloged procedure PL1DFC, and the effect of the statement

```
//PL1D.SYSPRINT DD SYSOUT=C
```

is to modify the existing DD statement SYSPRINT (causing the compiler listing to be transmitted to the system output device of class C).

Overriding DD statements must follow the EXEC statement that invokes the cataloged procedure in the same order as the corresponding DD statements of the cataloged procedure. DD statements that are being added must follow the overriding DD statements for the procedure step in which they are to appear. If you are using an operating system with PCP, an overriding or additional DD statement with the operand * or DATA must be the last DD statement for the procedure step.

To override a parameter of a DD statement, code either a revised form of the parameter or a replacement parameter that performs a similar function (e.g., SPLIT for SPACE). To nullify a parameter, code the keyword and equal sign without a value. You can override DCB subparameters by coding only those you wish to modify; that is, the DCB parameter in an overriding DD statement does not necessarily override the entire DCB parameter of the corresponding statement in the cataloged procedure.

Multiple Invocation of Cataloged Procedures

You can invoke different cataloged procedures, or invoke the same procedure several times, in the same job. No special problems are likely to arise unless more than one of these cataloged procedures involves a link-edit step, in which case you must take precautions to ensure that all your load modules can be executed.

The linkage editor always places a load module that it creates in the library identified by the DD statement SYSLMOD. In the absence of a linkage editor NAME statement (or the compiler OBJNM parameter option), it uses the member name specified in the DSNAME parameter as the name of the module. In the standard cataloged procedures, the DD statement SYSLMOD always specifies a temporary library named &&GOSET, and gives the member name GO.

Consider what will happen if, for example, you use the cataloged procedure PL1LFCLG twice in a job to compile, link-edit, and execute two PL/I programs, and do not individually name the two load modules that will be created by the linkage editor. The linkage editor will name the first module GO, as specified in the first DD statement SYSLMOD. It will not be able to use the same name for the second module, since the first module still exists in the library &&GOSET; therefore it will allocate a temporary name to the second module (a name that is not available to your program). Step GO of the cataloged procedure requests the job scheduler to initiate execution of the load module named in the DD statement SYSLMOD in the step LKED, that is, to execute the module named GO from the library &&GOSET. Consequently, the first program module will be executed twice and the second not at all.

You can use one of the following methods to obviate this difficulty:

1. Delete the library &&GOSET at the end of the step GO of the first invocation of the cataloged procedure by adding a DD statement of the form:

```
//GO.SYSLMOD DD DSNAME=&&GOSET,
//             DISP=(OLD,DELETE)
```

2. Modify the DD statement SYSLMOD in the second and subsequent invocations of the cataloged procedure so as to vary the names of the load modules. For example:

```
//LKED.SYSLMOD DD DSNAME=&&GOSET(GO1)
```

and so on.

3. Use the OBJNM option to give a different name to each load module.

Example

Figure 8-8 is an example of the use of the cataloged procedure PL1LFLG. It assumes that an object module already exists in the data set SOURCE1, which is on the 2311 disk pack with the serial number D186.

The PARM parameter in the EXEC statement nullifies the corresponding parameter of the first EXEC statement in the cataloged procedure; its effect is to suppress most of the linkage-editor listing.

Two of the DD statements in the example refer to the procedure step LKED; one of them overrides the existing statement SYSLMOD, and the other adds a new statement SYSIN. The overriding statement SYSLMOD causes the linkage editor to place the load module in the existing private library PLIB and give it the name CAT. The new statement SYSIN defines the data set that contains the object module that is to be link-edited. (The ddname SYSIN is equated with the name SYSLIN by the DDNAME parameter of the DD statement SYSLIN in the cataloged procedure.)

Data to be processed when the program CAT is executed is introduced in the input stream and is identified by the DD statement SYSIN, which is added to the job step GO.

```
//J070PGEX JOB
// EXEC PL1LFLG,PARM=
//LKED.SYSLMOD DD UNIT=2311,VOLUME=SER=D186,DSNAME=PLIB(CAT),
//             DISP=(OLD,PASS)
//LKED.SYSIN DD UNIT=2311,VOLUME=SER=D186,DSNAME=SOURCE1,DISP=OLD
//GO.SYSIN DD *
17324     259       7312      841       977       2193      21        37052
/*
```

Figure 8-8. Invoking Cataloged Procedure PL1LFLG

PERMANENT MODIFICATION

To make permanent modifications to a cataloged procedure, or to add a new cataloged procedure, use the system utility program IEBUPDTE, which is described in IBM System/360 Operating System: Utilities. The following paragraphs discuss some of the factors you should have in mind when considering whether to modify the standard cataloged procedures for your installation. For further information on writing installation cataloged procedures, see IBM System/360 Operating System: System Programmer's Guide.

In general, installation conventions will dictate the options that you include in the PARM, UNIT, and SPACE parameters of the cataloged procedures, and also the blocking factors for output data sets.

If your installation is using the MVT option of the operating system, you may need to modify some or all of the REGION parameters and SYSPRINT DD statements in the cataloged procedures.

The minimum region size for compilation should be at least 8K bytes larger than the largest value that will be specified in the compiler SIZE option (excluding SIZE=999999).

In those cataloged procedures that invoke the linkage editor, a region size of 96K is specified for the link-edit step. You can reduce this value if you are using the E-level linkage editor or the 44K F-level linkage editor. The minimum region sizes for the E-level linkage editor are:

Linkage Editor	Region
E15	24K
E18	26K
E44	54K

For the F-level linkage editor, the region size should be at least 8K bytes larger than the largest value to be specified in the SIZE option. You must alter the REGION parameter if you are using the 128K bytes, F-level, linkage editor.

Note that, under MVT the operating system requires 52K bytes of main storage within a region when initiating or terminating a job step. If you specify a region size of less than 52K bytes, completion of a job may be held up until 52K bytes are available.

If your installation does not use MVT, you can delete the REGION parameter from all cataloged procedures.

If the data set defined by a DD statement with the name SYSPRINT may be on a direct-access device, you should add a SPACE parameter to the statement.

A modified cataloged procedure can be tested by converting it to an in-stream procedure and executing it any number of times during a job. Figure 8-9 shows how to convert PL1DFC to an in-stream procedure and execute it twice.

For further information about in-stream procedures and symbolic parameters, refer to the publication IBM System/360 Operating System: Job Control Language Reference.

```
//CONVERT JOB PL1PROG,MSGLEVEL=1
//PL1COMPL PROC (Symbolic parameters are optional)
|
|   (PL1DFC procedure as it exists in the procedure library)
|
//ENDCOMPL PEND
//         EXEC PL1COMPL
//PL1D.SYSIN DD *
|
|       (Input Data)
|
/*
//         EXEC PL1COMPL
//PL1D.SYSIN DD *
|
|       (Input Data)
|
/*
```

Figure 8-9. Executing PL1DFC as an In-Stream Procedure

Chapter 9: Data Sets and PL/I Files

Introduction

This chapter describes briefly the nature and organization of data sets, and the data management services provided by IBM System/360 Operating System, and explains how the compiled program produced by the PL/I (F) compiler uses them.

Data Sets

In IBM System/360 Operating System, a data set is any collection of data that can be created and accessed by a program. A data set may be a deck of punched cards; it may be a series of items recorded on magnetic tape or paper tape; or it may be recorded on a direct-access device such as magnetic disk, drum, or data cell. A printed listing produced by a program is also a data set, but it cannot be read by a program. A PL/I compiled program uses the data management routines of the operating system to create and access data sets.

A data set resides on one or more volumes. A volume is a standard unit of auxiliary storage that can be written on or read by an input/output device (for example, a reel of tape, a disk pack, or a card deck); a unique serial number identifies each volume (other than a punched-card or paper-tape volume or a magnetic-tape volume with either no labels or nonstandard labels).

A magnetic-tape or direct-access volume can contain more than one data set; conversely, a single data set can span two or more magnetic-tape or direct-access volumes.

DATA SET NAMES

A data set on a direct-access device must have a name so that the operating system can refer to it. A data set on magnetic tape must have a name if the tape has standard labels (see 'Labels,' below). A name consists of up to eight characters, the first of which must be alphabetic. Data sets on punched cards, paper tape, unlabeled magnetic tape or nonstandard labeled magnetic tape do not have names.

You can place the name of a data set, with information identifying the volume on which it resides, in a catalog that exists in the volume that contains the operating system. Such a data set is termed a cataloged data set. To retrieve a cataloged data set, you do not need to give the volume serial number or identify the type of device; you need only specify the name of the data set and its disposition. The operating system searches the catalog for information associated with the name and uses this information to request the operator to mount the volume containing your data set.

If you have a set of related data sets, you can increase the efficiency of the search for a particular member of the set by establishing a hierarchy of indexes in the catalog. For example, consider an installation that groups its data sets under four headings: ENGRNG, SCIENCE, ACCNTS, and INVNTRY (Figure 9-1). In turn, each of these groups is subdivided; for instance, the SCIENCE group has subgroups called PHYSICS, CHEM, MATH, and BIOLOGY. The MATH group itself contains three subgroups: ALGEBRA, CALCULUS, and BOOL. To find the data set BOOL, the names of all the indexes of which it is part must be specified, beginning with the largest group (SCIENCE), followed by the next largest group (MATH), and finally the data set name BOOL. The names are separated by periods. The complete identification needed to find the data set BOOL is SCIENCE.MATH.BOOL; such an identifier is termed a qualified name. The maximum length of a qualified name is 44 characters, including the separating periods; each component name has a maximum length of eight characters. (Do not use data set names that begin with the letters SYS and have a P as the nineteenth character. The names assigned by the PCP operating system to unnamed temporary data sets are of this form. They are deleted when the operating system utility IEHPROGM is used with a SCRATCH statement that includes the keywords VTOC and SYS.)

Some data sets are updated periodically, or are logically part of a group of data sets, each of which is related to the others in time. You can relate such data sets to each other in what is termed a generation data group. Each data set in a generation data group has the same name qualified by a unique parenthesized generation number (for example, STOCK(0), STOCK(-1), STOCK(-2)). The most recently cataloged data set is generation 0, and the

preceding generations are -1, -2, and so on. You specify the number of generations to be saved when you establish the generation data group.

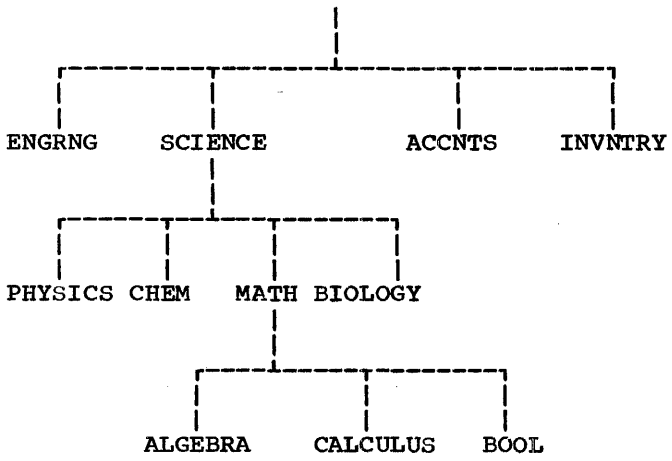


Figure 9-1. A Hierarchy of Indexes

For example, consider a generation data group that contains a series of data sets used for weather reporting and forecasting; the name of the data sets is WEATHER. The generations for the group (assuming that three generations are to be saved) are:

```
WEATHER(0)
WEATHER(-1)
WEATHER(-2)
```

When WEATHER is updated, the new data set is specified to the operating system as WEATHER(+1). When it catalogs the new data set, the operating system changes the name to WEATHER(0), changes the former WEATHER(0) to WEATHER(-1), the former WEATHER(-1) to WEATHER(-2), and deletes the former WEATHER(-2).

For instructions on how to create a generation data group, refer to IBM System/360 Operating System: Job Control Language and IBM System/360 Operating System: Utilities.

RECORD FORMATS

The items of data in a data set are arranged in blocks separated by interblock gaps (IBG)[1]; a block is the unit of data transmitted to and from a data set. Each block contains one record, part of a record or several records; a record is the unit of data transmitted to and from a program. When writing a PL/I program, you need consider only the records that you are reading or writing; but when you describe the data sets that your program will create or access, you must be aware of the relationship between blocks and records.

If a block contains two or more records, the records are said to be blocked. Blocking conserves storage space in a volume because it reduces the number of interblock gaps, and it may increase efficiency by reducing the number of input/output operations required to process a data set. Records are blocked and deblocked automatically with the aid of the data management routines of the operating system.

The records in a data set must be in one of three formats: fixed-length, variable-length, or undefined-length. Fixed-length and variable-length records can be blocked or unblocked; undefined-length records cannot be blocked. The following paragraphs describe the three record formats.

Fixed-Length Records

In a data set with fixed-length (F-format and FB-format) records, (see Figure 9-2) all records have the same length. If the records are blocked, each block contains an equal number of fixed-length records (although the last block may be truncated if there are insufficient records to fill it). If the records are unblocked, each record constitutes a block.

Because it can base blocking and deblocking on the constant record size, the operating system can process fixed-length records faster than variable-length records. The use of 'standard' FS-format and FBS-format further optimizes the sequential processing of a data set on a direct-access device. A standard format data set contains fixed-length records and must have no embedded empty tracks or short blocks (apart from the last block). With a standard format data set, the operating system can predict whether the next block of data will be on a new track, and, if necessary, can select a new read/write head in anticipation of the transmission of that block.

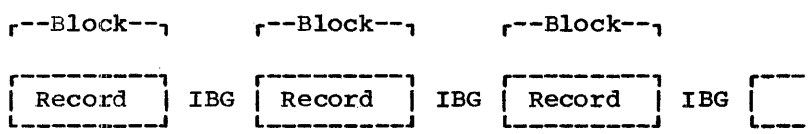
Note:

1. A PL/I program never places embedded short blocks in a data set with fixed-length records.

---

[1]Although the term 'interrecord gap' is widely used in operating system manuals, it is not used here; it has been replaced by the more accurate term 'interblock gap.'

Unblocked records (F-format):



Blocked records (FB-format):

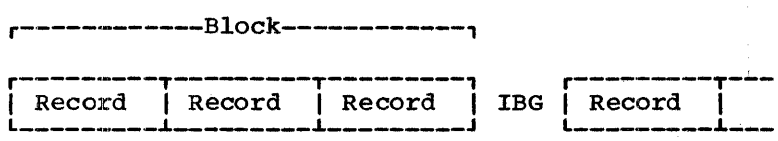


Figure 9-2. Fixed-Length Records

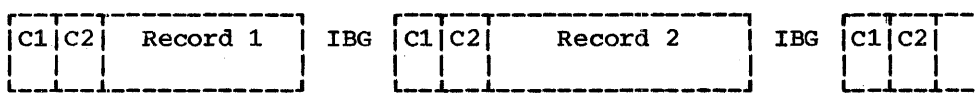
2. A data set can be processed as a standard data set even if it was not created as such, providing it contains no embedded short blocks or empty tracks.

Variable-Length Records

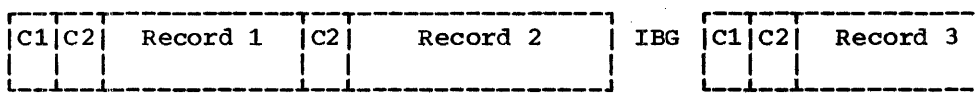
V-format permits both variable-length records and variable-length blocks. The first four bytes of each record and of each block contain control information for use by the operating system (including the length in bytes of the record or block). Variable-length records can have one of four formats: V, VB, VS, or VBS (Figure 9-3).

V-format signifies unblocked variable-length records. Each record is treated as a block containing only one record, the first four bytes of the block contain block control information, and the next four contain record control information.

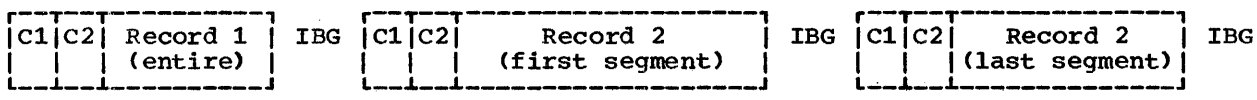
V-format:



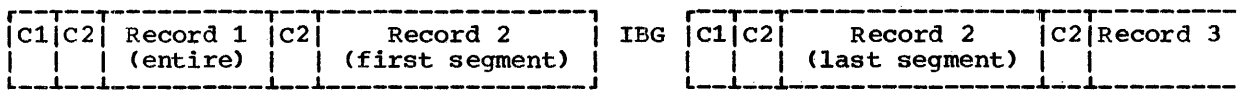
VB-format:



VS-format:



VBS-format:



C1: Block control information
C2: Record or segment control information

Figure 9-3. Variable-Length Records

VB-format signifies blocked variable-length records. Each block contains as many complete records as it can accommodate. The first four bytes of the block contain block control information, and the first four bytes of each record contain record control information.

VS-format is similar to V-format, but differs in that the length of a record can exceed the block length; if necessary, a record is segmented and continued in consecutive blocks. Each block contains only one record or segment of a record. The first four bytes of the block contain block control information, and the next four contain record or segment control information (including an indication of whether the record is complete or is a first, intermediate, or last segment).

VBS-format differs from VS-format in that each block contains as many complete records or segments as it can accommodate; each block is, therefore, approximately the same size (although there can be a variation of up to four bytes, since each segment must contain at least one byte of data).

VS-format and VBS-format records are known as spanned records because they can start in one block and be continued in the next. Segmentation and reassembly are handled automatically by the PL/I (F) compiler. The use of spanned records allows you to select a block size, independently of record size, that will combine optimum usage of auxiliary storage space with maximum efficiency of transmission. Note that spanned records cannot be specified for stream-oriented files.

Undefined-Length Records

U-format permits the processing of records that do not conform to F- and V-formats. The operating system and the PL/I (F) compiler treat each block as a record; your program must therefore perform any blocking or deblocking that you require.

DATA SET ORGANIZATION

The data management routines of the operating system can handle five types of data set, which differ in the way data is stored within them and in the permitted means of access to the data. Four of these types of data set, sequential, indexed sequential, direct, and telecommunications are recognized by the PL/I (F) compiler in the data set organizations CONSECUTIVE, INDEXED, and REGIONAL; and the file attribute TRANSIENT respectively; the fifth type, partitioned, has no corresponding PL/I organization[1].

In a sequential (or CONSECUTIVE) data set, records are placed in physical sequence. Thus, given one record, the location of the next record is determined by its physical position in the data set. Sequential organization is used for all magnetic tapes, and may be selected for direct-access devices. Paper tape, punched cards, and printed output are sequentially organized.

An indexed sequential (or INDEXED) data set must reside in a direct-access volume. Records are arranged in collating sequence, according to a key that is associated with every record. An index or set of indexes maintained by the operating system gives the location of certain principal records. This permits direct retrieval, replacement, addition, and deletion of records, as well as sequential processing.

A direct (REGIONAL) data set must be in a direct-access volume. The records within the data set can be organized by a PL/I program in three ways: REGIONAL(1), REGIONAL(2), and REGIONAL(3); in each case, the data set is divided into regions, each of which contains one or more records. A key that specifies the region number and, for REGIONAL(2) and REGIONAL(3), identifies the record, permits direct access to any record; sequential processing is also possible. There are no indexes.

A telecommunications data set (associated with a TRANSIENT file in a PL/I program) is an input or output message queue set up by the message control program. A key embedded in the record provides identification of the sending terminal.

In a partitioned data set, independent groups of sequentially organized data, each called a member, are stored in a direct-access volume. The data set includes a directory that lists the location of each member. Partitioned data sets are often called libraries.

---

[1]Do not confuse the operating system data set organizations 'sequential' and 'direct' with the PL/I file attributes SEQUENTIAL and DIRECT. The attributes refer to how the file is to be processed, and not to the way the corresponding data set is organized.

The PL/I (F) compiler includes no special facilities for creating and accessing partitioned data sets; however, there is ready access to the operating system facilities for partitioned data sets through job control language. Chapter 12, 'Libraries of Data Sets,' is a guide to the use of partitioned data sets for the PL/I programmer.

LABELS

The operating system uses recorded labels to identify magnetic-tape and direct-access volumes and the data sets they contain, and to store data set attributes (record size, block size, etc.). The attribute information must originally come from a DD statement or from your program. Once the label is written, however, you need not specify the information again.

Magnetic-tape volumes can have standard or nonstandard labels, or they can be unlabeled. Standard labels have two parts: the initial volume label, and header and trailer labels. The initial volume label identifies a volume and its owner; the header and trailer labels precede and follow each data set on the volume. Header labels contain system information, device-dependent information (for example, recording technique), and data-set characteristics. Trailer labels are almost identical with header labels, and are used when magnetic tape is read backwards.

Direct-access volumes have standard labels. Each volume is identified by a volume label, which is stored in a standard location in the volume. This label contains a volume serial number and the address of a volume table of contents (VTOC). The table of contents, in turn, contains a label (termed a data set control block (DSCB)) for each data set stored in the volume.

DATA DEFINITION (DD) STATEMENT

A data definition (DD) statement is a job control statement that describes a data set to the operating system, and is a request to the operating system, for the allocation of input/output resources. Each job step must include a DD statement for each data set that is processed by the step.

Chapter 1 describes the format of job control statements. The operand field of the DD statement can contain keyword parameters that describe the location of the data set (for example, volume serial number and identification of the unit on which the volume will be mounted) and the attributes of the data itself (record format, etc.).

The DD statement enables you to write PL/I source programs that are independent of the data sets and input/output devices they will use. You can modify the parameters of a data set or process different data sets without re-compiling your program; for example, you can modify a program that originally read punched cards so that it will accept input from magnetic tape merely by changing the DD statement.

Name of DD Statement

The name that appears in the name field of the DD statement (ddname) identifies the statement so that other job control statements and the PL/I program can refer to it. A ddname must be unique within a job step; if two DD statements in one job step have the same name, the second statement is ignored. With the two exceptions noted below, a DD statement must always have a name.

If the job step in which the DD statement appears is part of a cataloged procedure, the ddname must be qualified by the name of the procedure step. For example, a DD statement that describes a data set to be processed in step GO of the cataloged procedure PL1LFCLG might have the ddname GO.MSTR. (Note that the PL/I source program would refer to this DD statement only by its unqualified name MSTR.)

For input only you can concatenate two or more sequential or partitioned data sets (that is, link them so that they are processed as one continuous data set) by omitting the ddname from all but the first of the DD statements that describe them. For example, the following DD statements cause the data sets LIST1, LIST2, and LIST3 to be treated as a single data set for the duration of the job step in which the statements appear:

```
//GO.LIST DD DSNAME=LIST1,DISP=OLD
//         DD DSNAME=LIST2,DISP=OLD
//         DD DSNAME=LIST3,DISP=OLD
```

When read from a PL/I program the concatenated data sets need not be on the same volume, but the volumes must be on the same type of device, and the data sets must have similar characteristics (block size, record format, etc.). You cannot process concatenated data sets backwards.

Parameters of DD Statement

The operand field of the DD statement contains keyword parameters that you can use to give the following information:

1. The name of the data set (DSNAME parameter).
2. Description of the device and volume that contain the data set (UNIT, VOLUME, SPACE, LABEL, and SYSOUT parameters).
3. Disposition of the data set before and after execution of the job step (DISP parameter).
4. Data set characteristics (DCB parameter).

The following paragraphs summarize the functions of these groups of parameters. Appendix B describes the essential parameters and explains how to use them. For full details of all the parameters, refer to IBM System/360 Operating System: Job Control Language User's Guide, and Job Control Language Reference

Naming the Data Set

The DSNAME parameter specifies the name of a newly defined data set or refers to the name of an existing data set. You need not specify the DSNAME parameter for a temporary data set (one that exists only for the duration of the job step in which it is created); the operating system will give it a temporary name.

Describing the Device and Volume

The UNIT parameter specifies the type of input/output device to be allocated for the data set. You can specify the type by giving the actual unit address, the type number of the unit (e.g., 2400 for magnetic tape), or by naming a group of units established at system generation.

The VOLUME parameter identifies the volume on which the data set resides. It can also include instructions for mounting and demounting volumes.

The SPACE parameter specifies the amount of auxiliary storage required to accommodate a new data set on a direct-access device.

The LABEL parameter specifies the type and contents of the data set labels for magnetic tape.

The SYSOUT parameter allows you to route an output data set through a system output device. A system output device is any unit (but usually a printer or a card punch) that is used in common by all jobs. The computer operator allocates all the system output devices to specific classes according to device type and function. The usual convention is for class A to refer to a printer and class B to a card punch; the IBM-supplied cataloged procedures assume that this convention is followed. If you use the SYSOUT parameter, the only other information you may have to supply about the data set is the block size, which you can specify either in the DCB parameter or in your PL/I program.

Disposition of the Data Set

The DISP parameter indicates whether a data set already exists or is new, and specifies what is to be done with it at the end of the job step. At the end of a job step, you can delete a data set, pass it to the next step in the same job, enter its name in the system catalog or have it removed from the catalog, or you can retain the data set for future use without cataloging it.

The REWIND option of the ENVIRONMENT attribute allows you to use the DISP parameter to control the action taken when the end of a magnetic-tape volume is reached or when a magnetic-tape data set is closed. Refer to IBM System/360 Operating System: PL/I (F) Language Reference Manual for a description of the REWIND option.

Data Set Characteristics

The DCB (data control block) parameter of the DD statement allows you to describe the characteristics of the data in a data set, and the way it will be processed, at execution time. Whereas the other parameters of the DD statement deal chiefly with the identity, location, and disposal of the data set, the DCB parameter specifies information required for the processing of the records themselves. For DCB usage, see 'Data Control Block,' below.

The DCB parameter contains a list of subparameters that describe:

1. The organization of the data set and how it will be accessed (DSORG, OPTCD, CYLOFL, NCP, NTM, and LIMCT subparameters).
2. Device dependent information such as the recording technique for magnetic tape or the line spacing for a printer (CODE, DEN, MODE, PRTSP, STACK, and TRTCH subparameters).
3. The record format (BLKSIZE, LRECL, RECFM, KEYLEN, and RKP subparameters).

4. The number of data management buffers that are to be used (BUFNO subparameter).

5. The printer or card punch control characters (if any) that will be inserted in the first byte of each record (RECFM subparameter).

You can specify BLKSIZE, BUFNO, LRECL, NCP, RECFM, and TRKOFL in the ENVIRONMENT attribute of a file declaration in your PL/I program instead of in the DCB parameter.

You cannot use the DCB parameter to override information already established for the data set in your PL/I program (by the file attributes declared and the other attributes that are implied by them). DCB subparameters that attempt to change information already supplied are ignored. You can use only the DCB subparameters given above; if you specify any others, they will be ignored.

Data in the Input Stream

You can introduce data to your program by including it in the input job stream with your job control statements. The data must be in the form of 80-byte records (usually punched cards), and must be immediately preceded by a DD statement with the single parameter * in the operand field, for example:

```
//GO.SYSIN DD *
```

To indicate the end of the data, include the delimiter job control statement /*; this delimiter is not essential if you are using an operating system with MFT or MVT because the operating system will generate //SYSIN DD * statements for other data sets in the input stream.

If your data includes records that commence //, use the parameter DATA instead, for example:

```
//GO.SYSIN DD DATA
```

In this case, the delimiter (/*) is always necessary.

When using an operating system with PCP, you can include only one data set in the input stream for a job step, and the DD statement that defines it must be the last DD statement in the job step; for MFT and MVT, you can include more than one such data set in a job step. All three variants of the operating system supply full DCB subparameters for data sets in the input stream.

FILES AND DATA SETS

When you write a PL/I program, you do not need to know which data sets you will use or where the volumes that contain them will be mounted. PL/I uses a conceptual 'file' as a means of accessing a data set. When an OPEN statement is executed, the file is associated with a data set through the TITLE option, which refers to the name of the DD statement (ddname) that describes the data set; if the OPEN statement does not include the TITLE option, the compiler takes the DD name from the first eight characters of the file name, padding it with blanks if necessary.

The OPEN statement indicates the name of the DD statement that describes the data set to be associated with the file that is being opened; the DD statement specifies the type of device that will access the data set, the serial number of the volume that contains the data set, and the name of the data set (see Figure 9-4). If the DD statement refers to a cataloged data set, it need supply only the name of the data set and its disposition; the operating system can use the name to obtain unit and volume information from the system catalog.

Since the link between the PL/I file and the data set exists only while the file is open, the same file can be associated with different data sets during the execution of a single program; and the same data set can be accessed through different files. Furthermore, the use of a DD statement to define the data set, the volume that contains it, and the device on which they will be placed, enables you to defer your choice until execution time; and you can use the same program to process different data sets on different devices without re-compiling the program.

Operating System Data Management

The object program produced by the PL/I (F) compiler uses the data management facilities of the operating system to control the storage and retrieval of data. The compiler translates each input and output statement in a PL/I source program into a sequence of machine instructions that includes a branch to the appropriate PL/I library interface subroutine; this subroutine initiates the flow of control through any other PL/I library subroutines that are required. These subroutines issue

```
//SKID JOB
//  EXEC PL1LFCLG
//PL1L.SYSIN DD *
 PRINT:  PROC OPTIONS(MAIN);
         DCL  IN FILE RECORD SEQUENTIAL INPUT,
                 DATA CHAR(100),
                 LAB(3) LABEL INIT(A, B, C);
         DO I=1  TO  3;
            ON ENDFILE(IN) GO  TO  OUT;
            GO TO LAB(I);
            A:  OPEN FILE(IN) TITLE('T1');
               GO  TO  READ;
            B:  OPEN FILE(IN) TITLE('T2');
               GO  TO  READ;
            C:  OPEN FILE(IN) TITLE('T3');
            READ:  READ FILE(IN) INTO(DATA);
                   PUT FILE(SYSPRINT) SKIP LIST(DATA);
                   GO  TO  READ;
            OUT:  CLOSE FILE(IN);
                  END;
         END PRINT;
/*
//GO.T1 DD UNIT=2311, VOLUME=SER=1435, DSNAME=WINKEN, DISP=OLD
//GO.T2 DD UNIT=2311, VOLUME=SER=2374, DSNAME=BLINKEN, DISP=OLD
//GO.T3 DD UNIT=2311, VOLUME=SER=3412, DSNAME=NOD, DISP=OLD
```

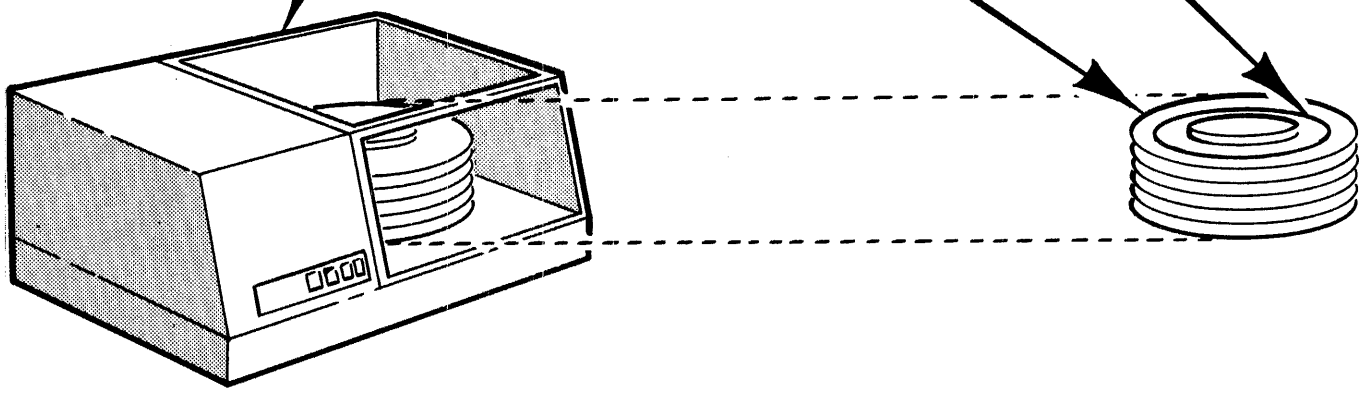


Figure 9-4. Associating a File with a Data Set

operating system assembler language macro instructions[1] that request the data management routines of the operating system to perform the required input or output operations. Most of the library subroutines required by a PL/I program are included with the object program in the load module produced by the linkage editor. However, some library subroutines, including those that handle the opening and closing of files, are loaded dynamically during the execution of the object program; the storage they occupy is released as soon as they have finished their work.

The operating system data management routines control the organization, location, and cataloging of data sets, as well as the storage and retrieval of the records they contain. They create and maintain data set labels, indexes, and catalogs; they move data between main storage and auxiliary storage; and they can use the system catalog to locate data sets and request the operator to mount and demount volumes as required.

BUFFERS

The data management routines can provide areas of main storage (buffers), in which data can be collected before it is transmitted to auxiliary storage, or into which it can be read before it is made available to a program. The employment of buffers permits the blocking and deblocking of records, and may allow the data management routines to increase the efficiency of transmission of data by anticipating the needs of a program. Anticipatory buffering requires at least two buffers: while the program is processing the data in one buffer, the next block of data can be read into another.

The operating system can further increase the efficiency of transmission in a program that involves many input/output operations by employing chained scheduling. In chained scheduling, a series of read or write operations are chained together and treated as a single operation. For chained scheduling to be effective, at least three buffers are necessary. (To specify chained scheduling, code OPTCD=C in the DCB subparameter of the DD statement: See Appendix B.)

The data management routines have two ways of making data that has been read into a buffer available to a program. In the move mode, the data is actually transferred from the buffer into the area of main storage occupied by the program. In the locate mode, the program can process the data while it is still in the buffer; the data management routines pass the address of the buffer to the program to enable it to locate the data. Similarly a program can move output data into the buffer or it can build the data in the buffer itself.

ACCESS METHODS

Data management has two techniques for transmitting data between main storage and auxiliary storage: the queued technique and the basic technique.

The queued access technique deals with individual records, which it blocks and deblocks automatically. A record is retrieved by the GET macro instruction and written by the PUT macro instruction. The first time a GET macro instruction is issued, the data management routines place a block of records in an input buffer and pass the first record to the program that issued the instruction; each succeeding GET passes another record to the program. When the input buffer is empty, it is automatically refilled with another block. Similarly, the PUT macro instruction places records in an output buffer and, when the buffer is full, writes out the records. Since the queued access technique brings records into main storage before they are requested, it can be used only for sequential processing.

The basic access technique uses the READ and WRITE macro instructions for input and output. These instructions move blocks, not records. When a READ macro instruction is issued, the data management routines pass a block of data to the program that issued the instruction; they do not deblock the records. Similarly, a WRITE macro instruction transmits a block to auxiliary storage. However, when the PL/I library subroutines employ the basic technique, they deblock the input records before passing them to the PL/I program, and block output records before issuing a WRITE macro instruction.

The combination of data-set organization and an access technique is termed an access method. The access methods used by the PL/I subroutine library are:

QSAM: Queued sequential access method

QISAM: Queued indexed sequential access method

---

[1]The macro instructions are described in IBM System/360 Operating System: Supervisor and ata Management Macro Instructions.

Data Set Organization	File Attributes			Access Methods
CONSECUTIVE	SEQUENTIAL	INPUT OUTPUT UPDATE	BUFFERED	QSAM
			UNBUFFERED	BSAM
INDEXED	SEQUENTIAL	INPUT OUTPUT UPDATE	BUFFERED or UNBUFFERED	QISAM
	DIRECT	INPUT UPDATE	-	BISAM
REGIONAL	SEQUENTIAL	INPUT UPDATE	BUFFERED or UNBUFFERED	QSAM/BSAM[1]
		OUTPUT		BSAM
	DIRECT	INPUT OUTPUT UPDATE	-	BDAM
TELEPROCESSING	TRANSIENT	INPUT OUTPUT	BUFFERED	QTAM

[1]QSAM is used for REGIONAL(1) BUFFERED but not KEYED.

Figure 9-5. Data Management Access Methods for Record-Oriented Transmission

BSAM: Basic sequential access method

BISAM: Basic indexed sequential access method

BDAM: Basic direct access method

QTAM: Queued telecommunications access method

The PL/I library subroutines use QSAM for all stream-oriented transmission. They implement PL/I GET and PUT statements by transferring the appropriate number of characters from or to the data management buffers, and use GET and PUT macro instructions in the locate mode to fill or empty the buffers. (For paper tape, however, the library uses the move mode to permit translation of the transmitted characters before passing them to the PL/I program.)

Figure 9-5 lists the access methods that the PL/I library subroutines use for record-oriented transmission.

Note that an access method identified with one data set organization can be used to process a data set usually thought of as organized in a different manner. For example, the PL/I (F) compiler uses BSAM and QSAM for the sequential processing of REGIONAL data sets.

DATA CONTROL BLOCK

A data control block (DCB) is an area of main storage that contains information about a data set and the volume that contains it. The data management routines of the operating system refer to this information when they are processing a data set; no data set can be processed unless there exists a corresponding DCB. For a PL/I program, a PL/I library subroutine creates a DCB for the associated data set when a file is opened.

A data control block contains two types of information: data set characteristics and processing requirements. The characteristics include record format, record size, block size, and data set organization. The processing information may specify the number of buffers to be used, and it may include device-dependent information (for example, printer line spacing or magnetic-tape recording density), and special processing options that are available for some data-set organizations.

The information in the DCB comes from three sources:

1. The file attributes declared implicitly or explicitly in the PL/I program. However, when link-editing more than one load or object module

together, take care that the first module to be included contains declarations of any files which are referred to but not declared in subsequent modules.

2. The data definition (DD) statement that describes the data set.

3. If the data set already exists, the data set labels.

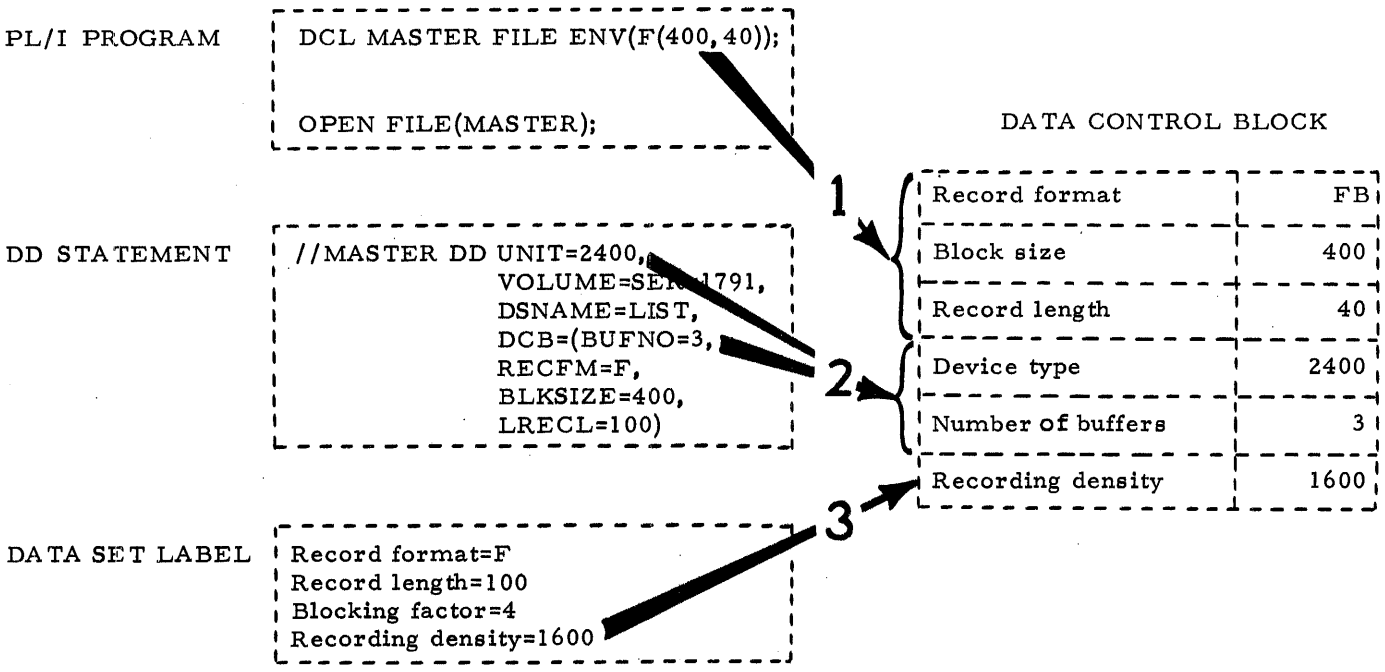
OPENING A FILE

The execution of a PL/I OPEN statement associates a file with a data set. This requires the merging of the information describing the file and the data set. If any conflict exists between file attributes and data set characteristics the UNDEFINEDFILE condition will be raised.

The data management subroutines of the PL/I library create a skeleton data control block for the data set, and use the file attributes from the DECLARE and OPEN statements, and any attributes implied by the declared attributes, to complete the data control block as far as possible (Figure 9-6). They then issue a data management OPEN macro instruction, which calls the operating system data management routines to check that the correct volume is mounted and to complete the data control block. The operating system routines examine the data control block to see what information is still needed and then look for this information, first in the DD statement, and finally, if the data set already exists and has standard labels, in the data set labels. For new data sets, the open routines begin to create the labels (if they are required) and to fill them with information from the data control block.

Neither the DD statement nor the data set label can override information provided by the PL/I program; nor can the data set label override information provided by the DD statement.

When the DCB fields have been filled in from these sources, control returns to the PL/I library subroutines. If any fields have still not been filled in, the PL/I OPEN subroutine provides default information for some of them; for example, if LRECL has not been specified, it is now provided from the value given for BLKSIZE.



Note: Note the order in which information from one source is overwritten by that from another.

Figure 9-6. How the Operating System completes the Data Control Block

CLOSING A FILE

The execution of a PL/I CLOSE statement dissociates a file from the data set with which it was associated. The PL/I library subroutines first issue a data management CLOSE macro instruction and then, when control returns from the operating system data management routines, release the data control block that was created when the file was opened. The data management routines complete the writing of labels for new data sets and update the labels of existing data sets.

Auxiliary Storage Devices

The following paragraphs state the record formats that are acceptable for various types of auxiliary storage device, and summarize the salient operational features of these devices.

CARD READER AND PUNCH

F-format, V-format, and U-format records are acceptable to both the card reader and punch; the control bytes of V-format records are not punched. Any attempt to block records is ignored.

Each punched card corresponds to one record; you should therefore restrict the maximum record size to 80 bytes (EBCDIC mode) or 160 bytes (column-binary mode). To select the mode, use the MODE subparameter of the DD statement; if you omit this subparameter, EBCDIC is assumed. (The column-binary mode increases the packing density of information on a card, punching two bytes in each column. Note that only six bits of each byte are punched; on input, the two high-order bits of each byte are set to zero; on output, the two high-order bits are lost.)

The Card Read Punch 2540 has five hoppers (stackers) into which cards are fed after reading or punching. Two stackers accept only cards that have been read, and two others accept only those that have been punched; the fifth (center) stacker can accept either cards that have been read or those that have been punched. The two stackers in each pair are numbered 1 and 2, and the center stacker is numbered 3 (Figure 9-7).

The Card Read Punch 2520 has two stackers, into which cards can be read or punched. The Card Reader 2501 has only one stacker.

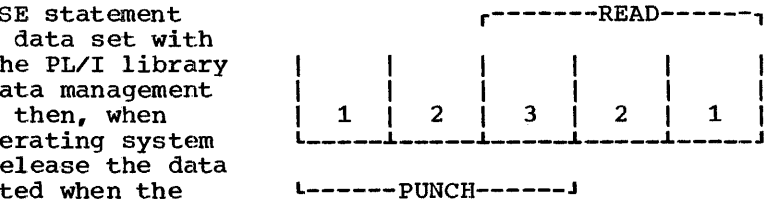


Figure 9-7. Card Read Punch 2540: Stacker Numbers

Cards are normally fed into the appropriate stacker 1 after reading or punching. You can use the STACK subparameter of the DD statement to select an alternative pocket for reading or punching. For punching only, you can select the stacker pocket dynamically by inserting a ANS or System/360 code in the first byte of each record; you must indicate which code you are using in the RECFM subparameter of the DD statement. The control character is not punched.

PAPER TAPE READER

The paper tape reader accepts F-format and U-format records; each U-format record is followed by an end-of-record character. You can use the CODE subparameter of the DD statement to request translation of data from one of the six standard paper tape codes to System/360 internal representation (EBCDIC). Any character found to have a parity error is not transmitted.

PRINTER

The printer accepts F-format, V-format, and U-format records; the control bytes of V-format records are not printed. Each line of print corresponds to one record; you should therefore restrict your record size to the length of one printed line. Any attempt to block records is ignored.

You can use the PRTSP subparameter of the DD statement to select the line spacing of your output, or you can control the spacing dynamically by inserting a ANS or System/360 code in the first byte of each record; you must indicate which code you are using in the RECFM subparameter of the DD statement. The control character is not printed. If you do not specify the line spacing, single spacing (no blanks between lines) is assumed.

MAGNETIC TAPE

Nine-track magnetic tape is standard in IBM System/360, but some 2400 series magnetic-tape drives incorporate features that facilitate reading and writing 7-track tape. The translation feature changes character data from EBCDIC (the 8-bit code used in System/360) to BCD (the 6-bit code used on 7-track tape) or vice-versa. The data conversion feature treats all data as if it were in the form of a bit string, breaking the string into groups of eight bits for reading into main storage, or into groups of six bits for writing on 7-track tape; the use of this feature precludes reading the tape backwards. If you want to employ translation or data conversion, include the TRTCH (tape recording technique) subparameter in your DD statement.

You can specify F-format, V-format, or U-format records for 9-track magnetic tape, but V-format records are acceptable on 7-track tape only if the data conversion feature is available. (The data in the control bytes of V-format records is in binary form; in the absence of the data conversion feature, only six of the eight bits in each byte are transmitted to 7-track tape.)

The maximum recording density available depends on the model number of the tape drive that you use; single-density tape drive units have a maximum recording density of 800 bytes per inch, and dual-density tape drive units have a maximum of 1600 bytes per inch. For 9-track tape, a single-density drive offers only the 800 bytes per inch density; the standard density for a dual-density drive is 1600 bytes per inch, but you can use the subparameter DEN (density) of the DD statement to specify 800 bytes per inch. For 7-track tape, the standard recording density for both types of drive unit is 800 bytes per inch; you can use the DEN subparameter to select alternatives of 200 or 556 bytes per inch.

Note: When a data check occurs on a magnetic-tape device with short length records (12 bytes on a read and 18 bytes on a write), these records will be treated as noise.

DIRECT-ACCESS DEVICES

Direct-access devices accept F-format, V-format, and U-format records.

The storage space on these devices is divided into conceptual cylinders and tracks. A cylinder is usually the amount of space that can be accessed without movement of the access mechanism, and a track is that part of a cylinder that is accessed by a single read/write head. For example, a 2311 disk pack has ten recording surfaces, each of which has 200 concentric tracks; thus, it contains 200 cylinders, each of which includes ten tracks.

When you create a new data set on a direct-access device, you must always indicate to the operating system how much auxiliary storage the data set will require. The way to do this is to use the SPACE parameter of the DD statement to allocate space in terms of blocks, tracks, or cylinders. If you request space in terms of tracks or cylinders, bear in mind that space in a data set on a direct-access device is occupied not only by blocks of data, but by control information inserted by the operating system; if you use small blocks, the control information can result in a considerable space overhead. The following reference cards contain tables that will enable you to determine the amount of space you will require:

2301 Drum Storage Unit, Form X20-1717

2302 Disk Storage Drive, Form X20-1706

2303 Drum Storage Unit, Form X20-1718

2311 Disk Storage Drive, Form X20-1705

2314 Storage Facility, Form X20-1710

2321 Data Cell Drive, Form X20-1704

Chapter 10: Stream-Oriented Transmission

Stream-oriented transmission allows a PL/I program to ignore block and record boundaries and treat a data set as a continuous stream of data items in character form. For output, the data management subroutines of the PL/I library convert the data items from the program variables into character form if necessary, and build the stream of characters into records for transmission to the data set. For input, the library subroutines take records from the data set and separate them into the data items requested by the program, converting them into the appropriate form for assignment to the program variables. Because stream-oriented transmission always treats the data in a data set as a continuous stream, it can be used only to process data sets with CONSECUTIVE organization.

Record Format

Although in stream-oriented transmission a PL/I program sees a data set as a continuous stream of data items, the data set actually comprises a series of discrete records in one of the three formats, fixed-length, variable-length, and undefined-length. Fixed-length and variable-length records can be blocked or unblocked; variable-length records, however, cannot be spanned.

When you create a data set, you must specify the record format either in your PL/I program or in a DD statement. In the PL/I program, you can state the record format directly in the ENVIRONMENT attribute, or you can indicate it indirectly in the LINESIZE option of an OPEN statement; the ENVIRONMENT attribute and the LINESIZE option are described in IBM System/360 Operating System: PL/I (F) Language Reference Manual. To specify the record format in a DD statement, use the DCB subparameters RECFM, BLKSIZE, and LRECL; these subparameters are described in Appendix B.

FIXED-LENGTH RECORDS

In a data set with fixed-length records, each record, except possibly the last, contains the same number of characters. Although the options and format items PAGE, LINE, and SKIP, and the format item COLUMN can cause a new output record (line) to be started, the current record is not truncated, but is padded to the specified length with blanks. The minimum length of a fixed-length record is 1 byte, except for magnetic tapes of density 800 per inch or less, in which case records of this length are treated as noise.

VARIABLE-LENGTH RECORDS

Variable-length output records are filled with characters up to the maximum length specified, unless they are truncated by the execution of one of the options or format items PAGE, LINE, SKIP, or COLUMN. If necessary, a variable-length record is padded with blanks to ensure that it contains the minimum ten characters of data that are required by the operating system. (With the eight bytes required for block and record control fields, this gives a minimum block size of 18 bytes.)

UNDEFINED-LENGTH RECORDS

Undefined-length records are processed in a similar manner to variable-length records, but, since the minimum length for these records is 1 byte, padding is never required. In the case of magnetic tapes of density 800 bytes per inch or less, 1-byte records are treated as noise.

CHOICE OF RECORD FORMAT

For a data set that is intended to be printed, you can usefully relate the record size to the length of the printed line (for example, by using the LINESIZE option); this is discussed under 'PRINT Files,' below. Similar considerations apply when you use the SKIP option or format item or the COLUMN format item with a non-PRINT file. For all other data sets that you create by stream-oriented transmission, program considerations do not affect the choice of record format (unless the data set will later be processed by record-oriented transmission). You will not normally gain any advantage by blocking your records; for unblocked records,

specify a block size, but no record size. Choose the maximum block size consistent with the amount of main storage available to your program and with the characteristics of the output device you are using. (The larger the block size, the fewer requests need be made to the data management routines for the transmission of a block of data to or from auxiliary storage; such requests carry a substantial time overhead.)

For a direct-access device, the ideal block size is the capacity of one track; this combines maximum transmission efficiency with optimum use of auxiliary storage space. A block can exceed the capacity of one track only if your installation includes the track overflow feature; this allows the block to be continued on the next track. Although track overflow can increase data-packing efficiency, it reduces transmission speed. The use of small blocks wastes storage space because a block of system information precedes each block of data on the track. For example, there are more than 60 bytes of system information for each block in the 2311 disk drive: a track can contain one block of 3625 bytes, but only 55 blocks of five bytes (i.e., 275 bytes of data). (Refer to the reference cards listed under 'Direct-Access Devices,' in Chapter 9, for further information.)

Unit record devices (card readers and punches, and printers) and paper-tape readers do not support blocked records. With a unit record device, each record begins on a new card or line regardless of the blocking factor. For example, if you specify LRECL=20, BLKSIZE=80 for a card punch, only the first 20 columns of each card will be punched; if you read these cards in, the stream will contain 60 blank characters between each of the original records. If the record size exceeds the capacity of a card (80 bytes) or the maximum length of a line (132 characters for the 1403 printer), the record is truncated.

In general, fixed-length records are transmitted faster than variable-length or undefined-length records. For maximum efficiency in sequential processing of a data set on a direct-access device, specify 'standard' format records (RECFM=FS or RECFM=FBS). If you use variable-length records, VBS-format allows you to specify block size independently of record size (and thus to select the optimum block size).

Buffers

Stream-oriented transmission requires the use of at least one data-management buffer, and ideally at least two buffers. You may gain some advantage by specifying three buffers for a low-speed device such as a printer, and you should specify at least three if you use chained scheduling. (For an explanation of chained scheduling, see Chapter 9 'Buffers'.)

You can request buffers either in your PL/I program (in the ENVIRONMENT attribute) or in a DD statement (BUFNO subparameter). If you omit the information or specify zero buffers, the operating system will allocate two buffers.

DCB Subparameters

Figure 11-2 lists the DCB subparameters that are applicable to a data set processed by stream-oriented transmission; they are described in Appendix B. You can specify record format, block size, record size, and number of buffers in the ENVIRONMENT attribute instead of in the DCB parameter.

Creating a Data Set

Any data set that you create using stream-oriented transmission must have CONSECUTIVE organization, but it is not necessary for you to specify this in the ENVIRONMENT attribute, since CONSECUTIVE is the default organization.

Your program deals only with data items, and not with records and blocks as they will exist in the data set. Accordingly, you need not concern yourself with the actual structure of the data set beyond specifying a block size (which is always necessary), unless you propose to use record-oriented transmission to access the data set at a later date.

To create a data set, you must give the operating system certain information either in the DD statement that defines the data set or in your PL/I program. The following paragraphs indicate the essential information, and discuss some of the optional information you may supply. Appendix B describes the parameters referred to, and explains how to code them; the ENVIRONMENT attribute and the LINESIZE option are discussed fully in IBM System/360 Operating System: PL/I (F) Language Reference Manual.

ESSENTIAL INFORMATION

You must supply the following information:

1. Device that will write or punch your data set (UNIT, SYSOUT, or VOLUME parameter of DD statement).

2. Block size: you can give the block size in the DD statement (BLKSIZE subparameter of DCB parameter), or in your PL/I program (in the ENVIRONMENT attribute or the LINESIZE option).

   Note: If you do not specify a record size, unblocked records are assumed and the record size is determined from the block size. If you do not specify a record format, U-format is assumed (except for PRINT files: see 'PRINT Files,' below).

If you want to keep a magnetic-tape or direct-access data set (that is, you do not want the operating system to delete it at the end of your job), the DD statement must name the data set and indicate how it is to be disposed of (DSNAME and DISP parameters). The DISP parameter alone will suffice if you want to use the data set in a later step but will not need it after the end of your job.

If you are creating a data set on a direct-access device, you must specify the amount of space required for it (SPACE parameter of DD statement).

If you want your data set written on a particular magnetic-tape or direct-access volume, you must indicate the volume serial number in the DD statement (SER or REF subparameter of VOLUME parameter). If you do not supply a serial number for a magnetic-tape data set that you want to keep, the operating system will allocate one and request the operator to print the number on your program listing.

If your data set is to follow another data set on a magnetic-tape volume, you must use the LABEL parameter of the DD statement to indicate its sequence number on the tape.

Figure 10-1 summarizes the essential parameters of the DD statements used for creating a data set with stream-oriented transmission.

Storage Device	Parameters of DD Statement		
	When required	What you must state	Parameters
All	Always	Output device	UNIT= or SYSOUT= or VOLUME=REF=
		Block size[1]	DCB=BLKSIZE=
Direct access only	Always	Storage space required	SPACE=
Magnetic tape only	Data set not first in volume and for magnetic tapes that do not have standard labels	Sequence number	LABEL=
Direct access and standard labeled magnetic tape	Data set to be used by another job step but is not required after end of job	Disposition	DISP=
	Data set to be kept after end of job	Disposition	DISP=
		Name of data set	DSNAME=
	Data set to be on particular volume	Volume serial number	VOLUME=SER= or VOLUME=REF=

[1]Alternatively, you can specify the block size in your PL/I program by using either the ENVIRONMENT attribute or the LINESIZE option.

Figure 10-1. Creating a Data Set: Essential Parameters of DD Statement

```
//J030PGEX JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 PEOPLE: PROC OPTIONS(MAIN);
         DCL WORK FILE STREAM OUTPUT,
             1 REC,
               2 FREC,
                 3 NAME CHAR(20),
                 3 NUM CHAR(1),
                 3 PAD CHAR(24),
               2 VREC CHAR(35),
             IN CHAR(80) DEF REC;
         ON ENDFILE(SYSIN) GO TO FINISH;
         OPEN FILE(WORK) LINESIZE(400);
 MORE:   GET FILE(SYSIN) EDIT(IN)(A(80));
         PUT FILE(WORK) EDIT(IN)(A(45+7*NUM));
         GO TO MORE;
 FINISH: CLOSE FILE(WORK);
         END PEOPLE;
/*
//GO.WORK DD UNIT=2311,SPACE=(400,10),DISP=(NEW,KEEP),DSNAME=PEOPLE,
//           VOLUME=SER=D186
//GO.SYSIN DD *
R.C.ANDERSON         0 202848 DOCTOR
B.F.BENNETT          2 771239 PLUMBER         VICTOR HAZEL
R.E.COLE             5 698635 COOK            ELLEN  VICTOR JOAN   ANN    OTTO
J.F.COOPER           5 418915 LAWYER          FRANK  CAROL  DONALD NORMAN BRENDA
A.J.CORNELL          3 237837 BARBER          ALBERT ERIC   JANET
E.F.FERRIS           4 158636 CARPENTER       GERALD ANNA   MARY   HAROLD
/*
```

Figure 10-2. Using Stream-Oriented Transmission to Create a Data Set

EXAMPLE

Figure 10-2 illustrates the use of stream-oriented transmission to create a data set on a 2311 disk drive. The data read from the input stream by the standard file SYSIN includes a field VREC that contains five unnamed 7-character subfields; the field NUM defines the number of these subfields that contain information. The output file WORK transmits to the data set the whole of the field FREC and only those subfields of VREC that contain information. The data set has U-format unblocked records (by default) with a maximum block size of 400 bytes (defined by the LINESIZE option in the statement that opens the file WORK). All blocks except the last will contain exactly 400 bytes.

Accessing a Data Set

A data set that you access using stream-oriented transmission need not have been created by stream-oriented transmission, but it must have CONSECUTIVE organization, and all the data in it must be in character form. You can open the associated file for input, and read the records the data set contains; or you can open the file for output, and extend the data set by adding records at the end.

To access an existing data set, you must identify it to the operating system in a DD statement. The following paragraphs indicate the essential information you must include in the DD statement, and discuss some of the other information you may supply; this discussion does not apply to data sets in the input stream, which are dealt with in Chapter 9 "Data Sets and PL/I Files". Appendix B describes the parameters referred to, and tells you how to code them.

ESSENTIAL INFORMATION

If the data set is cataloged, you need supply only the following information in the DD statement:

1. The name of the data set (DSNAME parameter). The operating system will locate the information that describes the data set in the system catalog, and, if necessary, will request the operator to mount the volume that contains it.

Parameters of DD Statement			
When required		What you must state	Parameters
Always		Name of data set	DSNAME=
		Disposition of data set	DISP=
If data set not cataloged	All devices	Input device	UNIT= or VOLUME=REF=
	Standard labeled magnetic tape and direct access	Volume serial number	VOLUME=SER=
Magnetic tape: if data set not first in volume or which does not have standard labels		Sequence number	LABEL=
If data set does not have standard labels		Block size[1]	DCB=BLKSIZE=
[1]Alternatively, you can specify the block size in your PL/I program by using either the ENVIRONMENT attribute or the LINESIZE option.			

Figure 10-3. Accessing a Data Set: Essential Parameters of DD Statement

2. Confirmation that the data set already exists (DISP parameter). If you are opening the data set for output with the intention of extending it by adding records at the end, code DISP=MOD; otherwise, to open the data set for output will result in it being overwritten.

If the data set is not cataloged, you must, in additon, specify the device that will read the data set and, for magnetic-tape and direct-access devices, give the serial number of the volume that contains the data set (UNIT and VOLUME parameters).

If the data set is on paper tape or punched cards, you must specify the block size either in the DD statement (BLKSIZE subparameter) or in your PL/I program (ENVIRONMENT attribute).

If the data set follows another data set on a magnetic-tape volume, you must use the LABEL parameter of the DD statement to indicate its sequence number on the tape.

Figure 10-3 summarizes the essential parameters of the DD statements for data sets accessed using stream-oriented transmission.

MAGNETIC TAPE WITHOUT STANDARD LABELS

If a magnetic-tape data set has nonstandard labels or is unlabeled, you must specify the block size either in the DD statement (BLKSIZE subparameter) or in your PL/I program (ENVIRONMENT attribute). But the DSNAME parameter is not essential if the data set is not cataloged.

PL/I data management includes no facilities for processing nonstandard labels. To the operating system, such labels appear as data sets preceding or following your data set. You can either process the labels as independent data sets or use the LABEL parameter of the DD statement to bypass them; to bypass the labels, code LABEL=(2,NL) or LABEL=(,BLP).

RECORD FORMAT

When you use stream-oriented transmission to read a data set, you do not need to know the record format of the data set (except when you must specify a block size); each GET statement transfers a discrete number of characters to your program from the data stream.

If you do give record-format information, it must be compatible with the actual structure of the data set. For example, if a data set was created with F-format records, a record size of 600 bytes, and a block size of 3600 bytes, you can access the records as if they were U-format with a maximum block size of 3600 bytes; but if you specify a block size of 3500 bytes, your data will be truncated.

```
//J033PGEX JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 PEOPLE: PROC OPTIONS(MAIN);
         DCL WORK FILE STREAM INPUT,
             1 REC,
               2 FREC,
                 3 NAME CHAR(20),
                 3 NUM CHAR(2),
                 3 SERNO CHAR(7),
                 3 PROF CHAR(16),
               2 VREC CHAR(35),
             IN CHAR(80) DEF REC;
         ON ENDFILE(WORK) GO  TO FINISH;
         OPEN FILE(WORK);
 MORE:   GET FILE(WORK) EDIT(IN,VREC)(A(45),A(7*NUM));
         PUT FILE(SYSPRINT) SKIP EDIT(REC)(A);
         GO TO MORE;
 FINISH: CLOSE FILE(WORK);
         END PEOPLE;
/*
//GO.WORK DD DSNAME=PEOPLE,DISP=(OLD,KEEP),UNIT=2311,VOLUME=SER=D186
```

Figure 10-4. Using Stream-Oriented Transmission to Access a Data Set

EXAMPLE

The program im Figure 10-4 reads the data set created in Figure 10-2 and uses the standard file SYSPRINT to list the data it contains. (SYSPRINT is discussed later in this chapter.) Each set of data is read, by the GET statement, into two variables: FREC, which always contains 45 characters; and VREC, which contains the number of characters generated by the expression 7*NUM. Note that the DISP parameter of the DD statement could read simply DISP=OLD; if the second term is omitted, an existing data set will not be deleted.

PRINT Files

Both the operating system and PL/I include features that facilitate the formatting of printed output. The operating system allows you to use the first byte of each record for a printer control character; the control characters, which are not printed, cause the printer to skip to a new line or page. In PL/I, the use of a PRINT file provides a convenient means of controlling the layout of printed output in stream-oriented transmission; the compiler automatically inserts printer control characters in response to the PAGE, SKIP, and LINE options and format items.

You can apply the PRINT attribute to any STREAM OUTPUT file, even if you do not intend to print the associated data set directly. When a PRINT file is associated with a magnetic-tape or direct-access data set, the control characters have no affect on the layout of the data set, but appear as part of the data in the records.

The compiler reserves the first byte of each record transmitted by a PRINT file for an ANS control character, and inserts the appropriate characters automatically. Figure 11-5 lists the ANS control characters; a PRINT file uses only the following five characters:

New Page	1
Single line space	b (blank)
Double line space	0
Triple line space	-
Suppress space	+

The compiler handles the PAGE, SKIP, and LINE options or format items by padding the remainder of the current record with blanks and inserting the appropriate control character in the next record. For V- or U-format files, this padding is not done if the current line length is equal to or greater than the length of a noise record. If SKIP or LINE requests more than a triple line space, the compiler inserts sufficient blank records with appropriate control characters to accomplish the required positioning. In the absence of a printer control option or format item, when a record is full the compiler inserts a blank code (single line space) in the first byte of the next record.

RECORD FORMAT

You can limit the length of the printed line produced by a PRINT file either by specifying a record size in the ENVIRONMENT attribute or in a DD statement, or by giving a line size in an OPEN statement. The record size must include the extra byte for the printer control character, that is, it must be one byte larger than the length of the printed line (five bytes larger for V-format records). The value you specify in the LINESIZE option refers to the number of characters in the printed line; the compiler adds the control bytes.

The blocking of records has no affect on the appearance of the output produced by a PRINT file, but it does result in more efficient use of storage space when the file is associated with a data set on magnetic tape or a direct-access device. If you use the LINESIZE option, ensure that your line size is compatible with your block size: for F-format records, blocksize must be an exact multiple of (line size + 1); for V-format records, blocksize must be at least nine bytes greater than line size.

Although you can vary the line size for a PRINT file during execution by closing the file and opening it again with a new line size, you must do so with caution if you are using the PRINT file to create a data set on magnetic tape or a direct-access device: you cannot change the record format established for the data set when the file is first opened. If the line size specified in an OPEN statement conflicts with the record format already established, the UNDEFINEDFILE condition will be raised; to prevent this, either specify V-format records with a block size at least nine bytes greater that the maximum line size you intend to use, or ensure that the first OPEN statement specifies the maximum line size. (Note that, if your program is processed by an operating system with MFT or MVT, output destined for the printer may be temporarily stored on a direct-access device, unless you specify a printer by using UNIT=, even if you intend it to be fed directly to the printer.)

Since PRINT files have a default line size of 120 characters, you need not give any record format information for them. In the absence of other information, the compiler assumes V-format records; the complete default information is:

```
BLKSIZE=129

LRECL=125

RECFM=VA
```

EXAMPLE

Figure 10-5 illustrates the use of a PRINT file and the printing options of the stream-oriented transmission statements to format a table and write it onto magnetic tape for printing on a later occasion. The table comprises the natural sines of the angles from 0° to 359° 54' in steps of 6'.

The statements in the ENDPAGE ON-unit insert a page number at the bottom of each page, and set up the headings for the following page. After the last line of the table has been written, the statement PUT FILE(TABLE) LINE(54) causes the ENDPAGE condition to be raised to ensure that a number appears at the foot of the last page; the preceding statement sets the flag FINISH to prevent a further set of headings from being written.

The DD statement that defines the data set that this program creates includes no record-format information; the compiler infers the following from the file declaration and the line size specified in the statement that opens the file TABLE:

```
Record format = V (the default for a
                   PRINT file)

Record size = 98 (line size + one byte
                  for printer control
                  character + four bytes
                  for record control
                  field)

Block size = 102 (record size + four
                  bytes for block
                  control field)
```

Figure 11-8 uses record-oriented transmission to print the table created in Figure 10-5.

TAB CONTROL TABLE

Data-directed and list-directed output to a PRINT file is automatically aligned on preset tabulator positions; the tab settings are stored in a table in the PL/I library module IHETAB (Figure 10-6). The functions of the fields in the table are as follows:

```
//J035PGEX JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 SINE:    PROC OPTIONS(MAIN);
          DCL TABLE FILE STREAM OUTPUT PRINT,
              TITLE CHAR(13) INIT('NATURAL SINES'),
              HEADINGS CHAR(90) INIT('         0         6        12        1
 8        24        30        36        42        48        54'),
              PGNO FIXED DEC(2) INIT(1),
              FINISH BIT(1) INIT('0'B),
              VALUES(0:359,0:9)FLOAT DEC(6);
          ON ENDPAGE(TABLE) BEGIN;
            PUT FILE(TABLE) EDIT('PAGE',PGNO)(LINE(55),COL(87),A,F(3));
            IF FINISH='0'B THEN DO;
            PGNO=PGNO+1;
            PUT FILE(TABLE) EDIT(TITLE||' (CONT''D)',HEADINGS)
                                 (PAGE,A,SKIP(3),A);
            PUT FILE(TABLE) SKIP(2);
            END;
            END;
          DO I=0 TO 359;
            DO J=0 TO 9;
              VALUES(I,J)=I+J/10;
              END;
            END;
          VALUES=SIND(VALUES);
          OPEN FILE(TABLE) PAGESIZE(52) LINESIZE(93);
          PUT FILE(TABLE) EDIT(TITLE,HEADINGS)(PAGE,A,SKIP(3),A);
          DO I=0 TO 71;
            PUT FILE(TABLE) SKIP(2);
            DO J=0 TO 4;
              K=5*I+J;
              PUT FILE(TABLE) EDIT(K,VALUES(K,*))(F(3),10 F(9,4));
              END;
            END;
          FINISH='1'B;
          PUT FILE(TABLE) LINE(54);
          CLOSE FILE(TABLE);
          END SINE;
/*
//GO.TABLE DD UNIT=2400,DISP=(NEW,CATLG,DELETE),DSNAME=SINES
```

Figure 10-5. Using a PRINT File

PAGESIZE	Halfword binary integer that defines the default page size.
LINESIZE	Halfword binary integer that defines the default line size.
Reserved bytes	Reserved for left and right margin facilities.
Tab count	Number of tab position entries in table (maximum 255). If tab count = 0, the tab positions are not used: each data item is put out as if a PRINT file were not being used.
Tab_1 - Tab_n	Tab positions within the print line. The first position is numbered 1, and the highest position is numbered 255. The value of each tab should be greater than that of the tab preceding it in the table; otherwise, it will be ignored. The first data field in the printed output begins at the left margin (position 1), and thereafter each field begins at the next available tab position.

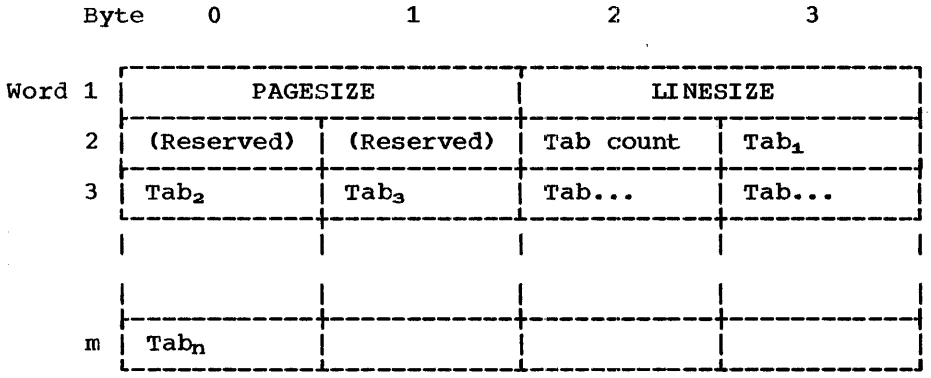
You can alter the tab control table and either replace it permanently in the PL/I library (SYS1.PL1LIB, see Chapter 12) or insert it in the linkage editor input stream for use in a single program. To change the tab settings, you must change the values in the assembler language control section (listed in Figure 10-6).

Figure 10-7 illustrates how to make a temporary change in the tab settings. The first job step (ASLE) uses the IBM-supplied assembler language cataloged procedure ASMFC to assemble a new control section for IHETAB, which retains the standard default line size and page size, but introduces new tab settings 30, 60, and 90. The second job step uses the cataloged procedure PL1LFCLG to compile, link edit, and execute a PL/I program. The procedure step LKED includes two additional DD statements: TAB, which defines the data set that contains the assembled control section (passed from step ASLE), and SYSIN, which permits the linkage editor INCLUDE statement to be inserted in the input stream. The INCLUDE statement causes the new tab control table (instead of the library module IHETAB) to be link edited with the PL/I object module.

Standard Files

PL/I includes two standard files, SYSIN for input and SYSPRINT for output. If your program includes a GET statement that does not include the FILE option, the compiler inserts the file name SYSIN; if it includes a PUT statement without the FILE option, the compiler inserts the name SYSPRINT.

If you do not declare SYSPRINT, the compiler will give the file the attribute PRINT in addition to the normal default



(a) Tab control table

60		120	
0	0	5	25
49	73	97	121

(b) Standard form of table

```
TAB       TITLE     'IHETAB'
IHETAB    CSECT
          ENTRY     IHETABS
IHETABS   DS        0D
PAGESIZE  DC        AL2(60)
LINESIZE  DC        AL2(120)
          DC        H'0'
NOTABS    DC        AL1(ENDTABS-*-1)
TAB1      DC        AL1(25)
TAB2      DC        AL1(49)
TAB3      DC        AL1(73)
TAB4      DC        AL1(97)
TAB5      DC        AL1(121)
ENDTABS   EQU       *
          END
```

(c) Control section IHETAB

Figure 10-6. Tabular Control Table (Module IHETAB)

```
//NEWTAB JOB
//ASLE EXEC ASMFC
//ASM.SYSPUNCH DD DSNAME=&&TAB,UNIT=SYSSQ,SPACE=(TRK,1),DISP=(NEW,PASS)
//ASM.SYSIN DD *
TAB       TITLE 'IHETAB'
IHETAB    CSECT
          ENTRY IHETABS
IHETABS   DS    0D
PAGESIZE  DC    AL2(60)
LINESIZE  DC    AL2(120)
          DC    H'0'
NOTABS    DC    AL1(ENDTABS-*-1)
TAB1      DC    AL1(30)
TAB2      DC    AL1(60)
TAB3      DC    AL1(90)
ENTABS    EQU   *
          END

/*
//COLEEX EXEC PL1LFCLG,PARM.LKED='LIST'
//PL1L.SYSIN DD *

Here follow the source statements of the PL/I program

/*
//LKED.TAB DD DSNAME=&&TAB,DISP=(OLD,DELETE)
//LKED.SYSIN DD *
 INCLUDE TAB
/*
//GO.SYSIN DD *

Here follows data for the PL/I program
/*
```

Figure 10-7. Making a Temporary Change in Tab Settings

attributes; thus, the complete file declaration will be SYSPRINT FILE STREAM OUTPUT PRINT EXTERNAL. Since SYSPRINT is a PRINT file, the compiler also supplies a default line size of 120 characters and a V-format record. Therefore, you need give only a minimum of information in the corresponding DD statement; if your installation uses the usual convention that the system output device of class A is a printer, the following is sufficient:

```
//SYSPRINT DD SYSOUT=A
```

If you use one of the IBM-supplied cataloged procedures to execute your program, even this DD statement is not required, since it is already included in the GO step.

You can override the attributes given to SYSPRINT by the compiler by explicitly declaring or opening the file. If you do so, bear in mind that this file is also used by the error-handling routines of the compiler, and that any change you make in the format of the output from SYSPRINT will also apply to the format of execution-time error messages. When an error message is printed, eight blanks are inserted at the start of each line except the first; consequently, if you specify a line size of less than nine characters (or a block size of less than ten bytes for F-format or U-format records, or less than 18 bytes for V-format records), the second and successive lines will not be printed, and the error-message routine will be locked in a permanent loop.

The compiler does not supply any special attributes for the standard input file SYSIN; if you do not declare it, it receives only the normal default attributes. The data set associated with SYSIN is usually in the input stream; if it is not in the input stream, you must supply full DD information.

Opening a STREAM File

Note that if a stream-oriented file is opened for output and closed without any PUT statements being executed for it, a blank record will be transmitted. If the file is a print file, the first byte of the record will contain a carriage control character to skip to a new page.

Chapter 11: Record-Oriented Transmission

In record-oriented transmission, data is transmitted to and from auxiliary storage exactly as it appears in the program variables; no data conversion takes place. A record in a data set corresponds to a variable in the program.

You can employ record-oriented transmission to process data sets with the three types of organization recognized by PL/I (CONSECUTIVE, INDEXED, and REGIONAL) and teleprocessing data sets. The creation and accessing of each type of data set are discussed under appropriate headings below.

Record Format

When you create a data set, you must specify a block size; you can also specify a record size. If you omit the record size, the records will be unblocked and their size will be determined by the block size. In your PL/I program, you can state the record format in the ENVIRONMENT attribute, which is described in IBM System/360 Operating System: PL/I (F) Language Reference Manual; alternatively, you can state the record format in a DD statement, using the subparameters RECFM, BLKSIZE, and LRECL (which are described in Appendix B).

With the following exceptions, record-oriented transmission can handle all three record formats (fixed-length, variable-length, and undefined-length). Fixed-length and variable-length records can be blocked or unblocked; variable-length records can be spanned. The exceptions are:

1. CONSECUTIVE data sets: you cannot read variable-length records backwards.
2. REGIONAL(1) and REGIONAL(2) data sets: only unblocked fixed-length records are accepted.
3. REGIONAL(3) data sets: you can use all three formats, but the records must be unblocked.

The record size (or block size if you omit the record size) that you should specify is governed by the size of the program variable from which you will transmit your data. For fixed-length records, the record size should equal the size of the variable; the maximum size of variable-length and undefined-length records must be enough to accept the largest record you will transmit. If these conditions are not met, the RECORD condition will be raised. To determine the record size required for a data aggregate, you may need to refer to the information on structure mapping included in IBM System/360 Operating System: PL/I (F) Language Reference Manual.

CHOICE OF RECORD FORMAT

For maximum efficiency in data transmission, use blocked records whenever they are permitted, and select the maximum block size consistent with the amount of main storage available to your program and with the characteristics of the output device you are using. (The larger the block size, the fewer requests need be made by the data management routines for the transmission of a block of data to or from auxiliary storage; such requests carry a substantial time overhead.)

For a direct-access device, the ideal block size is the capacity of one track; this combines maximum transmission efficiency with optimum use of auxiliary storage. A block can exceed the capacity of one track only if your installation has the track overflow feature; this allows the block to be continued on the next track. Although track overflow can increase data-packing efficiency, it reduces transmission speed. The use of small blocks wastes storage space because a block of system information follows each block of data (except the last on the track). For example, there are more than 60 bytes of system information for each block in the 2311 disk drive; a track can contain one block of 3625 bytes, but only 55 blocks of 5 bytes (i.e., 275 bytes of data). (Refer to the reference cards listed under 'Direct-Access Devices,' in Chapter 10, for further information.)

Unit record devices (card readers and punches, and printers) and paper-tape readers do not support blocked records. With a unit record device, each record begins on a new card or line regardless of the blocking factor. For example, if you specify LRECL=20, BLKSIZE=80 for a card punch, only the first 20 columns of each card will be punched. If the record size exceeds the capacity of a card (80 bytes)

or the maximum length of a line (132 characters for the 1403 printer), the record is truncated.

In general, fixed-length records are transmitted faster than variable-length or undefined-length records. For maximum efficiency in sequential processing of a data set on a direct-access device, specify 'standard' format records (RECFM=FS or RECFM=FBS). If you use variable-length records, VBS-format allows you to specify block size independently of record size (and thus to select the optimum block size).

Buffers

When you use a SEQUENTIAL BUFFERED file to process a data set, you can specify the number of data management buffers to be used either in your PL/I program (in the ENVIRONMENT attribute) or in the DD statement (BUFNO subparameter). If you omit this information or specify zero buffers, the operating system will allocate two buffers. Two buffers are sufficient for efficient transmission in most cases, but you may gain some advantage by specifying three buffers for a low-speed device such as a printer. You should specify at least three buffers if you use chained scheduling. Blocked records are not supported for UNBUFFERED files.

Although a buffer specification for a DIRECT file will be ignored, and even if the file is declared UNBUFFERED, the data management routines of the PL/I library sometimes use work buffers termed hidden buffers. Hidden buffers are used for:

1. UNBUFFERED access of CONSECUTIVE data sets with V-format records (the buffer contains the V-format control bytes and the data).
2. INDEXED data sets (the buffer contains a 10-byte control field and the data).
3. SEQUENTIAL access of a REGIONAL(2) or REGIONAL(3) data set declared KEYED (so that the key and the record can be transmitted from contiguous storage areas).

CREATING AND ACCESSING DATA SETS

To create or access a data set, you must give the operating system certain information, either in the DD statement that defines the data set or in your PL/I program. The following sections indicate the essential information that you must supply when processing CONSECUTIVE, INDEXED, REGIONAL, and teleprocessing data sets, and discuss some of the optional information you may supply. The discussions do not refer to data sets in the input stream, which are covered in Chapter 10. Appendix B describes the parameters referred to, and explains how to code them; the ENVIRONMENT attribute is described in IBM System/360 Operating System: PL/I (F) Language Reference Manual.

CONSECUTIVE Data Sets

CREATING A CONSECUTIVE DATA SET

In a CONSECUTIVE data set, records are stored sequentially in the order in which you write them; there are no keys. Figure 11-1 summarizes the essential information you must pass to the operating system when creating a CONSECUTIVE data set. You must specify:

1. Device that will write or punch your data set (UNIT, SYSOUT, or VOLUME parameter of DD statement).
2. Block size: you can give the block size in the DD statement (BLKSIZE subparameter of DCB parameter), or in your PL/I program (in the ENVIRONMENT attribute).

   Note: If you do not specify a record size, unblocked records are assumed and the record size is determined from the block size. If you do not specify a record format, U-format is assumed.

If you want to keep a magnetic-tape or direct-access data set (that is, you do not want the operating system to delete it at the end of your job), the DD statement must name the data set and indicate how it is to be disposed of (DSNAME and DISP parameters). The DISP parameter alone will suffice if you want to use the data set in more than one job step but will not need it after the end of your job.

If you are creating a data set on a direct-access device, you must specify the amount of space required for it (SPACE parameter of DD statement).

If you want your data set written on a particular magnetic-tape or direct-access volume, you must indicate the volume serial number in the DD statement (SER or REF subparameter of VOLUME parameter). If you do not supply a serial number of a

Storage Device	Parameters of DD Statement		
	When required	What you must state	Parameters
All	Always	Output device	UNIT= or SYSOUT= or VOLUME=REF=
		Block size[1]	DCB=BLKSIZE=
Direct access only	Always	Storage space required	SPACE=
Magnetic tape only	Data set not first in volume and for magnetic tapes that do not have standard labels	Sequence number	LABEL=
Direct access and standard labeled magnetic tape	Data set to be used by another job step but not required at end of job	Disposition	DISP=
	Data set to be kept after end of job	Disposition	DISP=
		Name of data set	DSNAME=
	Data set to be on particular volume	Volume serial number	VOLUME=SER= or VOLUME=REF=

[1]Alternatively, you can specify the block size in your PL/I program by using the ENVIRONMENT attribute.

Figure 11-1. Creating a CONSECUTIVE Data Set: Essential Parameters of DD Statement

magnetic-tape data set that you want to keep, the operating system will allocate one, inform the operator, and print the number on your program listing.

If your data set is to follow another data set on a magnetic-tape volume, you must use the LABEL parameter of the DD statement to indicate its sequence number on the tape.

Figure 11-2 lists the DCB subparameters that are applicable to CONSECUTIVE data sets; they are described in Appendix B. You can specify record format (RECFM), block size (BLKSIZE), record size (LRECL), and number of buffers (BUFNO) in the ENVIRONMENT attribute instead of in a DD statement.

Subparameter	Specifies
BLKSIZE	Maximum number of bytes per block
BUFNO	Number of data management buffers
CODE	Paper tape: code in which the tape was punched
DEN	Magnetic tape: tape recording density
LRECL	Maximum number of bytes per record
MODE	Card reader or punch: mode of operation (column binary or EBCDIC)
OPTCD	Optional data-management services and data-set attributes
PRTSP	Printer line spacing (0, 1, 2, or 3)
RECFM	Record format and characteristics
TRTCH	Magnetic tape: tape recording technique for 7-track tape

Figure 11-2. DCB Subparameters for CONSECUTIVE Data Sets

ACCESSING A CONSECUTIVE DATA SET

You can access an existing CONSECTUIVE data set in three ways. You can open the associated file for input, and read the records the data set contains; you can open the file for output, and extend the data

Parameters of DD Statement			
When required		What you must state	Parameters
Always		Name of data set	DSNAME=
		Disposition of data set	DISP=
If data set not cataloged	All devices	Input device	UNIT= or VOLUME=REF=
	Standard labeled magnetic tape and direct access	Volume serial number	VOLUME=SER=
Magnetic tape: if data set not first in volume or which does not have standard labels		Sequence number	LABEL=
If data set does not have standard labels		Block size[1]	DCB=BLKSIZE=
[1]Alternatively, you can specify the block size in your PL/I program by using the ENVIRONMENT attribute.			

Figure 11-3. Accessing a CONSECUTIVE Data Set: Essential Parameters of DD Statement

set by adding records at the end; or you can open the file for update, and read and rewrite each record in turn. (The operating system does not permit updating a CONSECUTIVE data set on magnetic tape; you must read the data set and write the updated records into a new data set.)

To access an existing data set, you must identify it to the operating system in a DD statement. The following paragraphs, which are summarized in Figure 11-3, indicate the essential information you must include in your DD statement, and discuss some of the other information you may supply; this discussion does not apply to data sets in the input stream, which are dealt with in Chapter 9 'Data Sets and PL/I Files'. Appendix B describes the DD parameters referred to, and tells you how to code them.

Essential Information

If the data set is cataloged, you need supply only the following information in the DD statement:

1. The name of the data set (DSNAME parameter). The operating system will locate the information that describes the data set in the system catalog, and, if necessary, will request the operator to mount the volume that contains it.
2. Confirmation that the data set already exists (DISP parameter). If you open the data set for output with the intention of extending it by adding records at the end, code DISP=MOD; otherwise, to open the data set for output will result in it being overwritten.

If the data set is not cataloged, you must, in addition, specify the device that will read the data set, and, for magnetic-tape and direct-access devices, give the serial number of the volume that contains the data set (UNIT and VOLUME parameters).

If the data set is on paper tape or punched cards, you must specify the block size either in the DD statement (BLKSIZE subparameter), or in your PL/I program (ENVIRONMENT attribute).

If the data set follows another data set on a magnetic-tape volume, you must use the LABEL parameter of the DD statement to indicate its sequence number on the tape.

Magnetic Tape Without Standard Labels

If a magnetic-tape data set has nonstandard labels or is unlabeled, you must specify the block size either in the DD statement (BLKSIZE subparameter) or in your PL/I program (ENVIRONMENT attribute). The DSNAME parameter is not essential if the data set is not cataloged.

PL/I data management includes no facilities for processing nonstandard labels. To the operating system, such labels appear as data sets preceding or following your data set. You can either process the labels as independent data sets or use the LABEL parameter of the DD statement to bypass them; to bypass the labels code LABEL=(2,NL) or LABEL=(,BLP).

Record Format

If you give record-format information, it must be compatible with the actual structure of the data set. For example, if a data set was created with F-format records, a record size of 600 bytes, and a block size of 3600 bytes, you can access the records as if they were U-format with a maximum block size of 3600 bytes; but if you specify a block size of 3500 bytes, your data will be truncated.

EXAMPLE OF CONSECUTIVE DATA SETS

Figure 11-4 illustrates both creation and accessing of CONSECUTIVE data sets on magnetic tape. The program merges the contents of two existing data sets, DS1 and DS2, and writes them onto a new data set, DS3; each of the original data sets contains 15-byte fixed-length records arranged in EBCDIC collating sequence. The two input files, IN1 and IN2, have the default attribute BUFFERED, and locate mode is used to read records from the associated data sets into the respective buffers. The output file, OUT, is not buffered, allowing move mode to be used to write the output records directly from the input buffers.

```
//J034PGEA JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 MERGE: PROC OPTIONS(MAIN);
        DCL  (IN1,IN2,OUT) FILE RECORD SEQUENTIAL,
             (ITEM1 BASED(A),ITEM2 BASED(B)) CHAR(15);
             ON ENDFILE(IN1) BEGIN;
               ON ENDFILE(IN2) GO TO FINISH;
 NEXT2:        WRITE FILE(OUT) FROM(ITEM2);
               READ FILE(IN2) SET(B);
               GO TO NEXT2;
               END;
             ON ENDFILE(IN2) BEGIN;
               ON ENDFILE(IN1) GO TO FINISH;
 NEXT1:        WRITE FILE(OUT) FROM(ITEM1);
               READ FILE(IN1) SET(A);
               GO TO NEXT1;
               END;
             OPEN FILE(IN1) INPUT,
                  FILE(IN2) INPUT,
                  FILE(OUT) OUTPUT;
             READ FILE(IN1) SET(A);
             READ FILE(IN2) SET(B);
 NEXT:       IF ITEM1>ITEM2 THEN DO;
               WRITE FILE(OUT) FROM(ITEM2);
               READ FILE(IN2) SET(B);
               GO TO NEXT;
               END;
             ELSE DO;
               WRITE FILE(OUT) FROM(ITEM1);
               READ FILE(IN1) SET(A);
               GO TO NEXT;
               END;
 FINISH:     CLOSE FILE(IN1),FILE(IN2),FILE(OUT);
             END MERGE;
/*
//GO.IN1   DD  DSNAME=DS1,DISP=(OLD,KEEP),UNIT=2400,VOLUME=SER=33731
//GO.IN2   DD  DSNAME=DS2,DISP=(OLD,KEEP),UNIT=2400,VOLUME=SER=987655
//GO.OUT   DD  DSNAME=DS3,DISP=(NEW,KEEP),UNIT=2400,
//              DCB=(RECFM=F,BLKSIZE=15)
```

Figure 11-4. Creating and Accessing a CONSECUTIVE Data Set

Code	Action
b	Space one line before printing (blank code)
0	Space two lines before printing
-	Space three lines before printing
+	Suppress space before printing
1	Skip to channel 1
2	Skip to channel 2
3	Skip to channel 3
4	Skip to channel 4
5	Skip to channel 5
6	Skip to channel 6
7	Skip to channel 7
8	Skip to channel 8
9	Skip to channel 9
A	Skip to channel 10
B	Skip to channel 11
C	Skip to channel 12
V	Select stacker 1
W	Select stacker 2

The channel numbers refer to the printer carriage control tape. (See IBM 1403 Printer Component Description.)

Figure 11-5. ANS Printer and Card Punch Control Characters

PRINTING AND PUNCHING CARDS

You cannot use a PRINT file for record-oriented transmission, and record-oriented transmission statements cannot include the printing options (PAGE, SKIP, etc). Nevertheless, you can still exercise some control over the layout of printed output by including a printer control code as the first byte of each of your output records; you can also use similar control codes to select the stacker to which cards punched by your program are fed.

The operating system recognizes two types of code for printer and card punch commands, ANS code and machine code. You must indicate which code you are using, either in the RECFM subparameter of your DD statement or in the ENVIRONMENT attribute in your PL/I program. If you specify one of these codes, but transmit your data to a device other than a printer or a card punch, the operating system will transmit the control bytes as part of your records. If you use an invalid control character, 'space one line' or 'stacker 1' will be assumed.

The ANS control characters (Figure 11-5) cause the specified action to occur before the associated record is printed or punched.

The machine code control characters differ according to the type of device. Figure 11-6 lists the codes for the 1403 Printer, and Figure 11-7 gives those for the 2540 Card Read Punch. There are two types of command for the printer, the one causing the action to occur after the record has been transmitted, and the other producing immediate action but transmitting

Print and then act	Action	Act immediately (no printing)
Code byte		Code byte
00000001	Print only (no space)	-
00001001	Space 1 line	00001011
00010001	Space 2 lines	00010011
00011001	Space 3 lines	00011011
10001001	Skip to channel 1	10001011
10010001	Skip to channel 2	10010011
10011001	Skip to channel 3	10011011
10100001	Skip to channel 4	10100011
10101001	Skip to channel 5	10101011
10110001	Skip to channel 6	10110011
10111001	Skip to channel 7	10111011
11000001	Skip to channel 8	11000011
11001001	Skip to channel 9	11001011
11010001	Skip to channel 10	11010011
11011001	Skip to channel 11	11011011
11100001	Skip to channel 12	11100011

The channel numbers refer to the printer carriage control tape. (See IBM 1403 Printer Component Description.)

Figure 11-6. 1403 Printer Control Codes

no data (i.e., you must include the second type of command in a blank record).

The essential requirements for producing printed output or punched cards are exactly the same as those for creating any other CONSECUTIVE data set (described above). For a printer, if you do not use one of the control codes, all data will be printed sequentially, with no spaces between records; each block will be interpreted as the start of a new line. When you specify a block size for a printer or card punch, and are using one of the control codes, include the control bytes in your block size; for example, if you want to print lines of 100 characters, specify a block size of 101.

Example

The program in Figure 11-8 uses record-oriented transmission to read and print the contents of the data set SINES, which was created by the PRINT file in Figure 10-5. Since the data set SINES was cataloged, only two parameters are required in the DD statement that defines it. The output file PRINTER is declared with the ENVIRONMENT option CTLASA, ensuring that the first byte of each record will be interpreted as an ANS printer control code. The information given in the ENVIRONMENT attribute could alternatively have been given in the DD statement as follows:

```
DCB=(RECFM=UA,BLKSIZE=94)
```

Code bytes	Action
00000001	Select stacker 1
01000001	Select stacker 2
10000001	Select stacker 3

Figure 11-7. 2540 Card Read Punch Control Codes

INDEXED Data Sets

A data set with INDEXED organization can exist only on a direct-access device. Each record in the data set is identified by a key that is recorded with the record. A key is a string of not more than 255 characters; all the keys in a data set must have the same length. The records in the data set are arranged according to the collating sequence of their keys. Once an INDEXED data set has been created, the keys facilitate the direct retrieval, addition, and deletion of records.

INDEXES

To provide faster access to the records in the data set, the operating system creates and maintains a system of indexes to the records in the data set. The lowest level of index is the track index. There is a track index for each cylinder in the data set; it occupies the first track (or tracks) of the cylinder, and lists the keys of the last records on each track in the cylinder. A search can then be directed to the first track that has a key that is higher than or equal to the key of the required record.

```
//J036PGEX JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 PRT:     PROC OPTIONS(MAIN);
          DCL TABLE FILE RECORD INPUT SEQUENTIAL,
              PRINTER FILE RECORD OUTPUT SEQUENTIAL ENV(V(102) CTLASA),
              LINE CHAR(94);
          ON ENDFILE(TABLE) GO TO FINISH;
 NEXT:    OPEN FILE(TABLE), FILE(PRINTER);
          READ FILE(TABLE) INTO(LINE);
          WRITE FILE(PRINTER) FROM(LINE);
          GO TO NEXT;
 FINISH:  CLOSE FILE(TABLE),FILE(PRINTER);
          END PRT;
/*
//GO.TABLE DD DSNAME=SINES,DISP=OLD
//GO.PRINTER DD SYSOUT=A
```

Figure 11-8. Printing with Record-Oriented Transmission

If the data set occupies more than one cylinder, the operating system develops a higher level index called a cylinder index. Each entry in the cylinder index identifies the key of the last record in the cylinder. To increase the speed of searching the cylinder index, you can request in a DD statement that the operating system develop a master index for a specified number of cylinders; you can have up to three levels of master index. Figure 11-9 illustrates the index structure. The part of the data set that contains the cylinder and master indexes is termed the index area.

When a INDEXED data set is created, all the records are written in what is called the prime data area. If more records are added later, the operating system does not rearrange the entire data set; it inserts each new record in the appropriate position and moves up the other records on the same track. Any records forced off the track by the insertion of a new record are placed in an overflow area. The overflow area can consist either of a number of tracks set aside in each cylinder for the overflow records from that cylinder (cylinder overflow area), or a separate area for all overflow records (independent overflow area).

Figure 11-10 shows how the records in the overflow area are chained together and to the track index so as to maintain the logical sequence of the data set. Each entry in the track index consists of two parts:

1. The normal entry, which points to the last record on the track.

2. The overflow entry, which contains the key of the first record transferred to the overflow area and also points to the last record transferred from the track to the overflow area.

If there are no overflow records from the track, both index entries point to the last record on the track. An additional field is added to each record that is placed in the overflow area. It points to the previous record transferred from the same track; the first record from each track is linked to the corresponding overflow entry in the track index.

CREATING AN INDEXED DATA SET

When you create an INDEXED data set, your program must write the records in the data set sequentially in the order of ascending key values; the associated file must be opened for SEQUENTIAL OUTPUT.

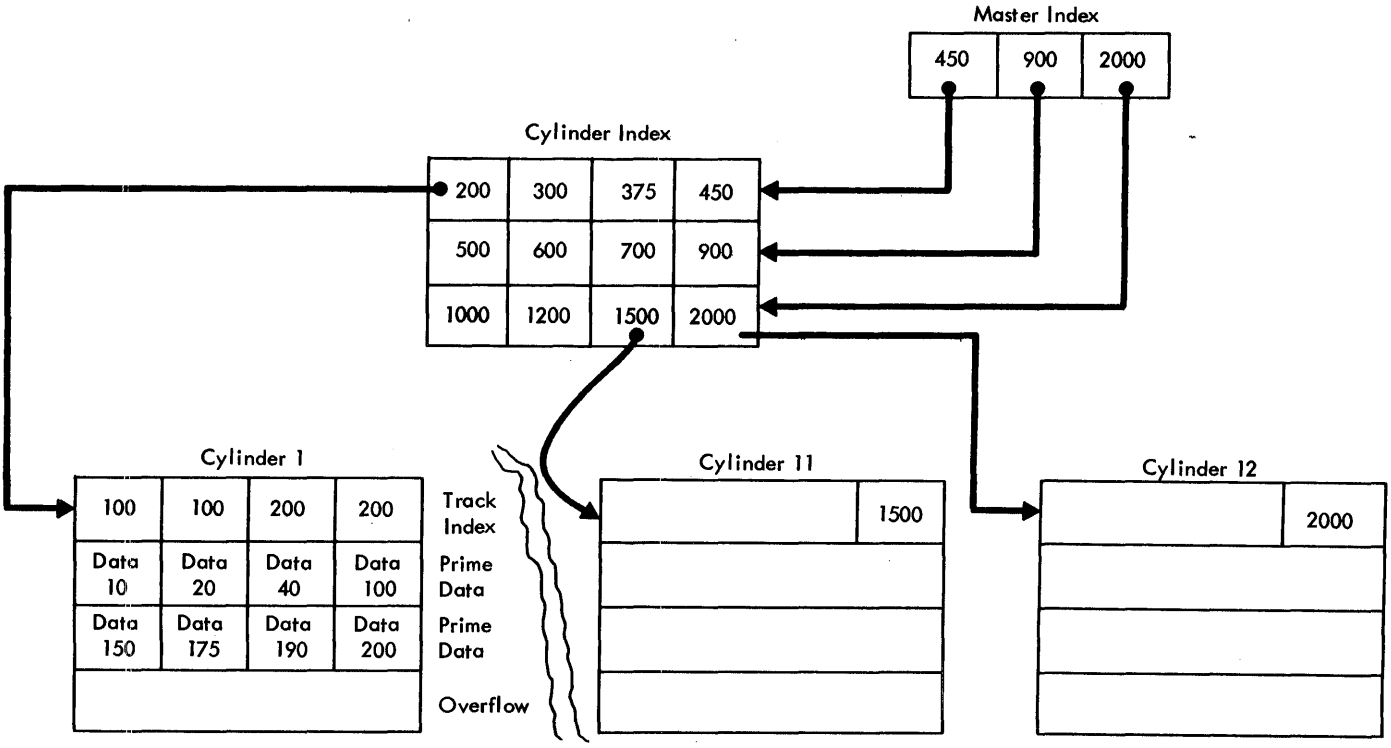


Figure 11-9. Index Structure of INDEXED Data Set

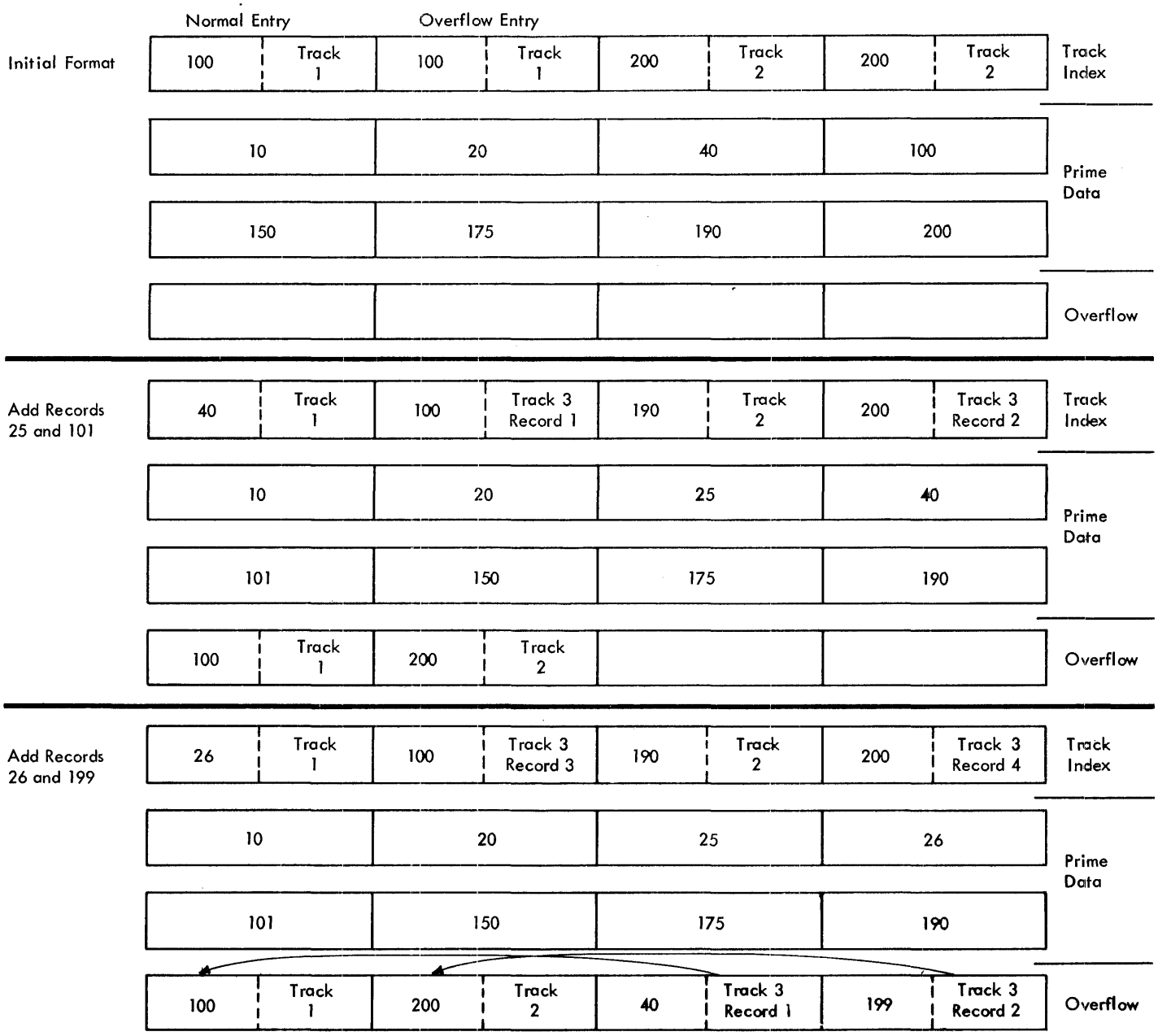


Figure 11-10. Adding Records to an INDEXED Data Set

Once an INDEXED data set has been created, it can be extended if the file is reopened for SEQUENTIAL OUTPUT. However, it is your responsibility to ensure that the key of the first record to be added to the data set is higher than the highest key already contained in the data set. Failure to ensure this will cause diagnostic message IHE031I to be printed and the program will terminate abnormally with a system completion code of 031.

You can use a single DD statement to define the whole of the data set (index area, prime area, and overflow area), or you can use two or three statements to define the areas independently. If you use two DD statements, you can define either the index area and the prime area together, or the prime area and the overflow area together.

If you want the whole of the data set to be on a single volume, there is no advantage to be gained by using more than one DD statement except to define an independent overflow area (see 'Overflow Area,' below). But, if you use separate DD statements to define the index and/or overflow areas on volumes separate from

that which contains the prime area, you will increase the speed of direct access to the records in the data set by reducing the number of access mechanism movements required.

When you use two or three DD statements to define an INDEXED data set, the statements must appear in the order: index area; prime area; overflow area. The first DD statement must have a name (ddname), but the name fields of a second or third DD statement must be blank. The DD statements for the prime and overflow areas must specify the same type of unit (UNIT parameter). You must include all the DCB information for the data set in the first DD statement; DCB=DSORG=IS will suffice in the other statements.

Essential Information

In general, all the information given above for the creation of a CONSECUTIVE data set on a direct-access device applies equally to an INDEXED data set. The following paragraphs discuss only the constraints imposed by the use of INDEXED organization and the additional information you must supply or may want to give. Figure 11-11 summarizes all the essential parameters required in a DD statement for the creation of an INDEXED data set, and Figure 11-12 lists the DCB subparameters needed.

Appendix B contains a description of all the parameters of the DD statement.

You cannot place an INDEXED data set on a system output (SYSOUT) device.

You must request space for the prime data area in the SPACE parameter. Your request must be in units of cylinders unless you place the data set in a specific position on the volume (by specifying a track number in the SPACE parameter). In the latter case, the number of tracks you specify must be equivalent to an integral number of cylinders, and the first track must be the first track of a cylinder other than the first cylinder in the volume. You can also use the SPACE parameter to specify the amount of space to be used for the cylinder and master indexes (unless you use a separate DD statement for this purpose). If you do not specify the space for the indexes, the operating system will use part of the independent overflow area; if there is no independent overflow area, it will use part of the prime data area.

In the DCB parameter, you must always state the data set organization (DSORG=IS), and in the first (or only) DD statement you must also give the length of the keys (KEYLEN).

Parameters of DD Statement		
When required	What you must state	Parameters
Always	Output device	UNIT= or VOLUME=REF=
	Storage space required	SPACE=
	Data control block information: refer to Figure 11.12	DCB=
More than one DD statement	Name of data set and area (index, prime, overflow)	DSNAME=
Data set to be used in another job step but not required after end of job	Disposition	DISP=
Data set to be kept after end of job	Disposition	DISP=
	Name of data set	DSNAME=
Data set to be on particular volume	Volume serial number	VOLUME=SER= or VOLUME=REF=

Figure 11-11. Creating an INDEXED Data Set: Essential Parameters of DD Statement

Name of Data Set

If you use only one DD statement to define your data set, you need not name the data set unless you intend to access it in another job. But, if you include two or three DD statements, you must specify a data set name, even for a temporary data set.

The DSNAME parameter in a DD statement that defines an INDEXED data set not only gives the data set a name, but it also identifies the area of the data set to which the DD statement refers:

DSNAME=name(INDEX)

DSNAME=name(PRIME)

DSNAME=name(OVFLOW)

If the data set is temporary, prefix its name with &&. If you use one DD statement to define the prime and index or prime and overflow areas, code DSNAME=name(PRIME); if you use only one DD statement, code DSNAME=name(PRIME), or simply DSNAME=name.

Record Format and Keys

An INDEXED data set can contain both fixed- and variable-length records, blocked or unblocked. You must always include the subparameter RECFM in your DD statement or specify the record format in your PL/I program (ENVIRONMENT attribute).

The key associated with each record can be contiguous with or embedded within the data in the record; you can save storage space in the data set if you use blocked records with embedded keys.

DCB Subparameters

	To specify	Subparameters
These are always required	Record format[1]	RECFM=F,FB,FBS, V[2],or VB[2]
	Block size[1]	BLKSIZE=
	Data set organization	DSORG=IS
	Key length	KEYLEN=
Include at least one of these if overflow is required	Cylinder overflow area and number of tracks per cylinder for overflow records	OPTCD=Y and CYLOFL=
	Independent overflow area	OPTCD=I
These are optional	Record length[1]	LRECL=
	Embedded key (relative key position)	RKP=
	Master index	OPTCD=M
	Automatic processing of dummy records	OPTCD=L
	Number of data management buffers[1]	BUFNO=
	Number of tracks in cylinder index for each master index entry	NTM=

[1]Alternatively, can be specified in ENVIRONMENT attribute.

[2]For V or VB format records, it is essential that RKP is specified with a value that is equal to or greater than 4.

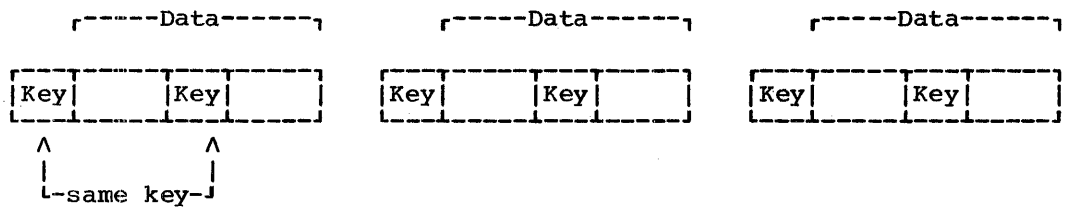
Note: Full DCB information must appear in the first, or only, DD statement. Subsequent statements require only DSORG=IS.

• Figure 11-12. DCB Subparameters for INDEXED Data Set

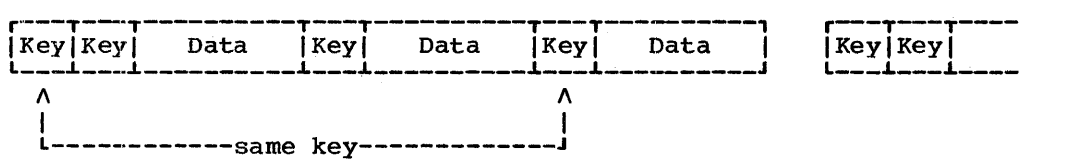
If the records are unblocked, the key of each record is recorded in the data set in front of the record even if it is also embedded within the record (Figure 11-13, (a) and (b)). If blocked records do not have embedded keys, the key of each record is recorded within the block in front of the record, and the key of the last record in the block is also recorded in front of the block (Figure 11-13(c)). When blocked records have embedded keys, the individual keys are not recorded separately in front of each record in the block; the key of the last record in the block is recorded in front of the block (Figure 11-13(d)).

(a) Unblocked records, non-embedded keys

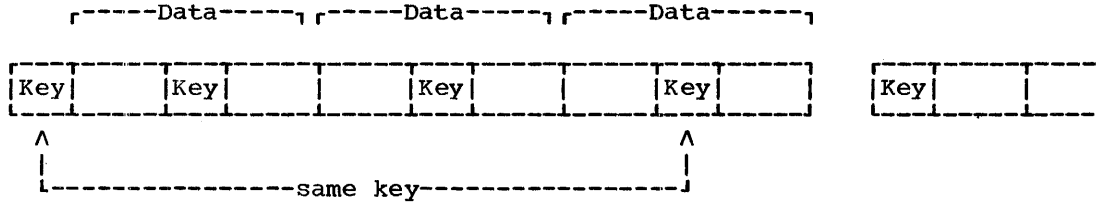
(b) Unblocked records, embedded keys



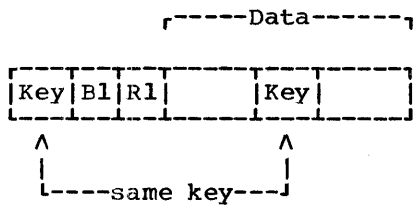
(c) Blocked records, non-embedded keys



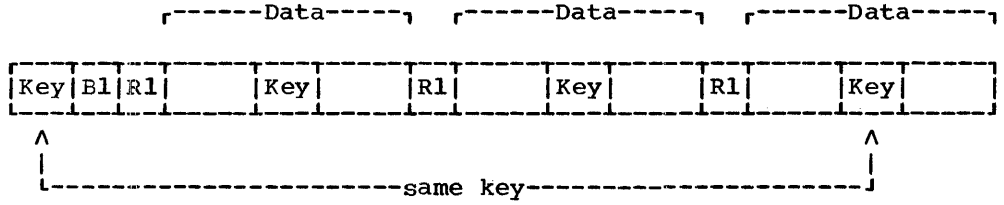
(d) Blocked records, embedded keys



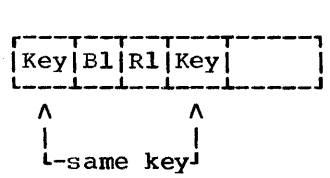
(e) Unblocked variable length records, RKP>4



(f) Blocked variable length records, RKP>4



(g) Unblocked variable length records, RKP=4



(h) Blocked variable length records, RKP=4

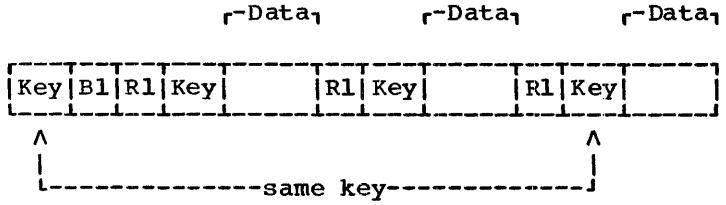


Figure 11-13. Record Formats in an INDEXED Data Set

If you use blocked records with non-embedded keys, the record size that you specify must include the length of the key, and the block size must be a multiple of this combined length. Otherwise, record size and block size refer only to the data in the record. (See Figure 11-14.)

If you use records with embedded keys, you must include the DCB subparameter RKP to indicate the position of the key within the record. For fixed-length records the value specified in the RKP subparameter is one less than the byte number of the first character of the key; that is, if RKP=1, the key starts in the second byte of the record. The value assumed if you omit this subparameter is RKP=0, which specifies that the key is not embedded in the record but is separate from it.

For variable-length records, the value specified in the RKP subparameter must be the relative position of the key within the record plus four. The extra four bytes takes into account the 4-byte control field used with variable-length records. For this reason you must never specify RKP less than four. When deleting records you must always specify RKP equal to or greater than five, since the first byte of the data is used to indicate deletion.

Note: For unblocked records, the key, even if embedded, is always recorded in a position preceding the actual data.

	RKP	LRECL	BLKSIZE
Blocked records	Not zero	R	R * B
	Zero or omitted	R + K	B * (R+K)
Unblocked records	Not zero	R	R
	Zero or omitted	R	R
V-Blocked records	>4	≥R+4	≥1*(R+4)+4
V-Unblocked records	>4	≥R+4	≥R+8
V-Blocked records	=4	≥R+K+4	≥1*(R+K+4)+4
V-Unblocked records	=4	≥R+K+4	≥R+K+8

R = Size of data in record

K = Length of keys (as specified in KEYLEN subparameter)

B = Blocking factor

Example: For blocked records, non-embedded keys, 100 bytes of data per record, 10 records per block, key length = 20: LRECL=120,BLKSIZE=1200,RECFM=FB

Figure 11-14. Record Format Information for an INDEXED Data Set

Consequently, the RKP subparameter need not be specified for fixed length unblocked records as the default RKP value is 0. Under these circumstances the key is only

recorded once in the block. For variable length records the minimum RKP value is 4 and the RKP must always be specified.

Overflow Area

If you intend to add records to the data set on a future occasion, you must request either a cylinder overflow area or an independent overflow area, or both.

For a cylinder overflow area, include the DCB subparameter OPTCD=Y and use the subparameter CYLOFL to specify the number of tracks in each cylinder to be reserved for overflow records. A cylinder overflow area has the advantage of a short search time for overflow records, but the amount of space available for overflow records is limited, and much of the space may be unused if the overflow records are not evenly distributed throughout the data set.

For an independent overflow area, use the DCB subparameter OPTCD=I to indicate that overflow records are to be placed in an area reserved for overflow records from all cylinders, and include a separate DD statement to define the overflow area. The use of an independent overflow area has the advantage of reducing the amount of unused space for overflow records, but entails an increased search time for overflow records.

It is good practice to request cylinder overflow areas large enough to contain a reasonable number of additional records and an independent overflow area to be used as the cylinder overflow areas are filled.

If the prime data area is not filled during creation, you cannot use the unused portion for overflow records, nor for any records subsequently added during direct access (although you can fill the unfilled portion of the last track used). You can reserve space for later use within the prime data area by writing 'dummy' records during creation: see 'Dummy Records,' below.

Master Index

If you want the operating system to create a master index for you, include the DCB subparameter OPTCD=M, and indicate in the NTM subparameter the number of tracks in the cylinder index you wish to be referred to by each entry in the master index. The operating system will automatically create up to three levels of master index, the first two levels addressing tracks in the next lower level of master index.

Dummy Records

You cannot change the specification of an INDEXED data set after you have created it. Therefore, you must foresee your future needs where the size and location of the index, prime, and overflow areas are concerned, and you must decide whether you want the operating system to identify and skip dummy (deleted) records.

If you code OPTCD=L, the operating system will flag any record that is named in a DELETE statement by placing the bit string (8)'1'B in the first byte. Subsequently, during SEQUENTIAL processing of the data set, such records will be ignored; if they are forced off a track when the data set is being updated, they will not be placed in the overflow area. Do not specify OPTCD=L when you are using blocked records with non-embedded keys; if you do, the string (8)'1'B will overwrite the key of the 'deleted' record.

You can include a dummy record in an INDEXED data set by setting the first byte of data to (8)'1'B and writing the record in the usual way.

ACCESSING AN INDEXED DATA SET

You can open an existing INDEXED data set for sequential or direct access, and for input or update in each case. Sequential input allows you to read the records in ascending key sequence, and in sequential update you can read and rewrite each record in turn; during sequential access, if OPTCD=L was specified when the data set was created, dummy records are ignored. Using direct input, you can read records using the READ statement, and in direct update you can read or delete existing records or add new ones.

Note that only one DIRECT UPDATE file should be open at any one time to add records to an INDEXED data set. Further, if two files are open simultaneously, one for sequential and one for direct processing of the same INDEXED data set, some records might become inaccessible to the SEQUENTIAL file due to changes to track indexes made for the DIRECT file when it adds records to the data set. The records that may be inaccessible are those added to the INDEXED data set in the following ways:

1. Records added to the end of the data set.

2. Records written directly into the overflow area of the data set.

Parameters of DD Statement		
When required	What you must state	Parameters
Always	Name of data set	DSNAME=
	Disposition of data set	DISP=
	Data Set Organization	DCB=DSORG=IS
If data set not cataloged	Input device	UNIT= or VOLUME=REF=
	Volume serial number	VOLUME=SER=

• Figure 11-15. Accessing an INDEXED Data Set: Essential Parameters of DD Statement

3. Records written on the overflow area when forced out of the prime data area by records being added to the prime data area.

To access an existing INDEXED data set, you must identify it to the operating system in one, two or three DD statements; the DD statements must correspond with those used when the data set was created. The following paragraphs indicate the essential information you must include in each DD statement, and Figure 11-15 summarizes this information. Appendix B describes the parameters referred to, and tells you how to code them.

If the data set is cataloged, you need supply only the following information in each DD statement:

1. The name of the data set (DSNAME parameter). The operating system will locate the information that describes the data set in the system catalog and, if necessary, will request the operator to mount the volume that contains it.

2. Confirmation that the data set already exists (DISP parameter).

3. Confirmation that the data set organization is INDEXED (DSORG subparameter of the DCB parameter).

If the data set is not cataloged, you must, in addition, specify the device that will process the data set and give the serial number of the volume that contains it (UNIT and VOLUME parameters).

Note: If you add a new record to a data set whose overflow areas are already full, the new record will not be added to the data set and the file will remain unchanged; the KEY condition will be raised. To reduce the likelihood of this occurrence, ensure that your overflow areas are sufficiently large, and reorganize the data set at regular intervals (see below).

REORGANIZING AN INDEXED DATA SET

It is necessary to reorganize an INDEXED data set periodically because the addition of records to the data set results in an increasing number of records in the overflow area. Therefore, even if the overflow area does not eventually become full, the average time required for the direct retrieval of a record will increase. The frequency of reorganization depends on how often the data set is updated, on how much storage is available in the data set, and on your timing requirements.

Reorganizing the data set also eliminates records that are marked as 'deleted,' but are still present within the data set.

There are two ways to reorganize an INDEXED data set:

1. Read the data set into an area of main storage or onto a temporary CONSECUTIVE data set, and then recreate it in the original area of auxiliary storage.

2. Read the data set sequentially and write it into a new area of auxiliary storage; you can then release the original auxiliary storage.

EXAMPLES OF INDEXED DATA SETS

Figure 11-16 illustrates the creation of a simple INDEXED data set. The data set contains a telephone directory, using the subscribers' names as keys to the telephone numbers.

```
//J002PGEX JOB
//CREATE EXEC PL1LFCLG,PARM.LKED='LIST'
//PL1L.SYSIN DD *
 TELNOS: PROC OPTIONS(MAIN);
         DCL DIREC FILE RECORD SEQUENTIAL KEYED ENV(INDEXED),
             CARD CHAR(80),
             NAME CHAR(20) DEF CARD,
             NUMBER CHAR(3) DEF CARD POS(21),
             IOFIELD CHAR(3);
         ON ENDFILE(SYSIN) GO TO FINISH;
         OPEN FILE(DIREC) OUTPUT;
 NEXTIN: GET FILE(SYSIN) EDIT(CARD)(A(80));
         IOFIELD=NUMBER;
         WRITE FILE(DIREC) FROM(IOFIELD) KEYFROM(NAME);
         GO TO NEXTIN;
 FINISH: CLOSE FILE(DIREC);
         END TELNOS;
/*
//GO.DIREC DD UNIT=2311,SPACE=(CYL,1),DCB=(RECFM=F,BLKSIZE=3,DSORG=IS,
//             KEYLEN=20,OPTCD=LIY,CYLOFL=2),DSNAME=TELNO(INDEX),
//             DISP=(NEW,KEEP),VOLUME=SER=D186
//          DD UNIT=2311,SPACE=(CYL,4),DCB=DSORG=IS,DSNAME=TELNO(PRIME),
//             DISP=(NEW,KEEP),VOLUME=SER=D186
//          DD UNIT=2311,SPACE=(CYL,4),DCB=DSORG=IS,
//             DSNAME=TELNO(OVFLOW),DISP=(NEW,KEEP),VOLUME=SER=D186
//GO.SYSIN DD *
ACTION,G.          162
BAKER,R.           152
BRAMLEY,O.H.       248
CHEESEMAN,L.       141
CORY,G.            336
ELLIOTT,D.         875
FIGGINS,S.         413
HARVEY,C.D.W.      205
HASTINGS,G.M.      391
KENDALL,J.G.       294
LANCASTER,W.R.     624
MILES,R.           233
NEWMAN,M.W.        450
PITT,W.H.          515
ROLF,D.E.          114
SHEERS,C.D.        241
SUTCLIFFE,M.       472
TAYLOR,G.C.        407
WILTON,L.W.        404
WINSTONE,E.M.      307
/*
```

Figure 11-16. Creating an INDEXED Data Set

The program in Figure 11-17 updates this data set and prints out its new contents. The input data includes codes to indicate the operations required:

A: Add a new record
C: Change an existing record
D: Delete an existing record

REGIONAL Data Sets

A data set with REGIONAL organization can exist only on a direct-access device. A REGIONAL data set is divided into regions that are numbered consecutively from zero. The following paragraphs briefly describe the three types of REGIONAL organization.

In a REGIONAL(1) data set, a region is a record. Each record in the data set is identified by its region number, an unsigned decimal integer not exceeding 16777215. Region numbers start from 0 at the beginning of the data set. There are no recorded keys.

```
//J041PGEX JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 DIRUPDT:PROC OPTIONS(MAIN);
         DCL DIREC FILE RECORD KEYED ENV(INDEXED),
             NUMBER CHAR(3),
             NAME CHAR(20),
             CODE CHAR(1);
         ON ENDFILE(SYSIN) GO TO PRINT;
         ON KEY(DIREC) BEGIN;
           IF ONCODE=51 THEN PUT FILE(SYSPRINT) SKIP EDIT
                             ('NOT FOUND:',NAME)(A(15),A);
           IF ONCODE=52 THEN PUT FILE(SYSPRINT) SKIP EDIT
                             ('DUPLICATE:',NAME)(A(15),A);
           END;
         OPEN FILE(DIREC) DIRECT UPDATE;
 NEXT:   GET FILE(SYSIN) EDIT(NAME,NUMBER,CODE)(A(20),A(3),X(56),A(1));
         IF CODE='A' THEN WRITE FILE(DIREC) FROM(NUMBER) KEYFROM(NAME);
         ELSE IF  CODE='C' THEN REWRITE FILE(DIREC) FROM(NUMBER)
                                                        KEY(NAME);
              ELSE IF CODE='D' THEN DELETE FILE(DIREC) KEY(NAME);
                   ELSE PUT FILE(SYSPRINT) SKIP EDIT('INVALID CODE:',
                                             NAME)(A(15),A);
         GO TO  NEXT;
 PRINT:  CLOSE FILE(DIREC);
         PUT FILE(SYSPRINT) PAGE;
         OPEN FILE(DIREC) SEQUENTIAL INPUT;
         ON ENDFILE(DIREC) GO TO FINISH;
 NEXTIN: READ FILE(DIREC) INTO(NUMBER) KEYTO(NAME);
         PUT FILE(SYSPRINT) SKIP EDIT(NAME,NUMBER)(A);
         GO TO NEXTIN;
 FINISH: CLOSE FILE(DIREC);
         END DIRUPDT;
/*
//GO.DIREC DD DSNAME=TELNO(INDEX),DISP=(OLD,KEEP),UNIT=2311,
//            VOLUME=SER=D186,DCB=DSORG=IS
//         DD DSNAME=TELNO(PRIME),DISP=(OLD,KEEP),UNIT=2311,
//            VOLUME=SER=D186,DCB=DSORG=IS
//         DD DSNAME=TELNO(OVFLOW),DISP=(OLD,KEEP),UNIT=2311,
//            VOLUME=SER=D186.DCB=DSORG=IS
//GO.SYSIN DD *
NEWMAN,M.W.          516450                                          C
GOODFELLOW,D.T.      889                                             A
MILES,R.                233                                          D
HARVEY,C.D.W.        209                                             A
BARTLETT,S.G.        183                                             A
CORY,G.                 336                                          D
READ,K.M.            001                                             A
PITT,W.H.               515
ROLF,D.F.               114                                          D
ELLIOTT,D.           291875                                          C
HASTINGS,G.M.           391                                          D
BRAMLEY,O.H.         439248                                          C
/*
```

• Figure 11-17. Updating an INDEXED Data Set

REGIONAL(2) organization is similar to REGIONAL(1), but differs in that a key is recorded with each record. The record key is a string of not more than 255 characters. A record is written in the first vacant space after the beginning of the track that contains the region number specified in the WRITE statement; for retrieval, the search for a record begins on the track that contains the region number specified in the READ statement, and may continue through the data set until the record has been found.

A REGIONAL(3) data set is similar in organization and in operation to a REGIONAL(2) data set, with the difference that each region corresponds to one track of the direct-access device and is not a record position. Therefore, depending on the record size, a region can contain one or more records.

The major advantage of REGIONAL organization over other types of data set organization is that it allows you to control the relative placement of records; by judicious programming, you can optimize record access in terms of device capabilities and the requirements of particular applications. REGIONAL(1) organization is most suited to applications where there will be no duplicate region numbers, and where most of the regions will be filled (obviating wasted space in the data set). REGIONAL(2) and REGIONAL(3) are more appropriate where records are identified by numbers that are thinly distributed over a wide range. You can include in your program an algorithm that derives the region number from the number that identifies a record in such a manner as to optimize the use of space within the data set; duplicate region numbers will occur, but their only effect might be to lengthen the search time for records with duplicate region numbers.

REGIONAL(1) and REGIONAL(2) data sets can contain only F-format unblocked records, but a REGIONAL(3) data set can have unblocked records of all three formats, F, V, and U. The examples at the end of this section illustrate typical applications of all three types of REGIONAL organization.

CREATING A REGIONAL DATA SET

You can use either sequential or direct-access to create a REGIONAL data set.

In sequential creation, you must present records in order of ascending region numbers; for REGIONAL(1) and REGIONAL(2) the region number for each record must exceed that of the preceding record since each region can contain only one record. In all cases, dummy records (identified by (8)'1'B in the first byte) are placed automatically in regions whose numbers are skipped.

For direct creation, one of the PL/I library subroutines formats the whole of the data set when you open the corresponding file. For REGIONAL(1) and (2), and for REGIONAL(3) with F-format records, formatting involves filling the data set with dummy records; for REGIONAL(3) with U-format or V-format records, a record, called the capacity record, is written at the start of each track to indicate an empty track. During creation, you can present records in any order.

Essential Information

In general, all the information given above for the creation of a CONSECUTIVE data set on a direct-access device applies equally to a REGIONAL data set. The following paragraphs discuss only the constraints imposed by the use of REGIONAL organization and the additional information you must supply or may want to give. Figure 11-18 summarizes all the essential parameters required in a DD statement for the creation of a REGIONAL data set, and Figure 11-19 lists the DCB subparameters you will need to use. Appendix B contains a description of all the parameters of the DD statement.

You cannot place a REGIONAL data set on a system output (SYSOUT) device.

Parameters of DD Statement		
When required	What you must state	Parameters
Always	Output device	UNIT= or VOLUME=REF=
	Storage space required	SPACE=
	Data control block information: refer to Figure 11-19	DCB=
Data set to be used in another job step but not required in another job	Disposition	DISP=
Data set to be kept after end of job	Disposition	DISP=
	Name of data set	DSNAME=
Data set to be on particular volume	Volume serial number	VOLUME=SER= or VOLUME=REF=

Figure 11-18. Creating a REGIONAL Data Set: Essential Parameters of DD Statement

In the DCB parameter, you must always state the data set organization (DSORG=DA). For REGIONAL(2) and REGIONAL(3), you must also state the length of the recorded key (KEYLEN): refer to IBM System/360 Operating System: PL/I (F) Language Reference Manual for a description of how the recorded key is derived from the source key supplied in the KEYFROM option.

For REGIONAL(2) and REGIONAL(3), if you want to restrict the search for space to add a new record, or the search for an existing record, to a limited number of tracks beyond the track that contains the specified region, use the LIMCT subparameter of the DCB parameter. If you omit this parameter, the search will continue to the end of the data set, and then from the beginning back to the starting point.

DCB Subparameters		
	To specify	Subparameters
These are always required	Record format[1]	RECFM=F or RECFM=V or RECFM=U REGIONAL(3) only
	Block size[1]	BLKSIZE=
	Data set organization	DSORG=DA
	Key length (REGIONAL(2) and (3) only)	KEYLEN=
These are optional	Limited search for a record or space to add a record (REGIONAL(2) and (3) only)	LIMCT=
	Number of data management buffers[1]	BUFNO=
[1]Alternatively, can be specified in ENVIRONMENT attribute.		

Figure 11-19. DCB Subparameters for REGIONAL Data Set

ACCESSING A REGIONAL DATA SET

You can open an existing REGIONAL data set for sequential or direct access, and for input or update in each case. Using sequential input with a REGIONAL(1) data set you can read all the records in ascending region-number sequence, and in sequential update you can read and rewrite each record in turn. Sequential access of a REGIONAL(2) or REGIONAL(3) data set will give you the records in the order in which they appear in the data set, which is not necessarily region-number order. Using direct input, you can read any record by supplying its region number and, for REGIONAL(2) and REGIONAL(3), its recorded key; in direct update, you can read or delete existing records or add new ones. The operating system ignores dummy records in a REGIONAL(2) or REGIONAL(3) data set; but a program that processes a REGIONAL(1) data set must be prepared to recognize dummy records.

To access a REGIONAL data set, you must identify it to the operating system in a DD statement. The following paragraphs indicate the minimum information you must include in the DD statement; they are summarized in Figure 11-20. Appendix B describes the parameters referred to and explains how to code them.

If the data set is cataloged, you need supply only the following information in your DD statement:

1. The name of the data set (DSNAME parameter). The operating system will locate the information that describes the data set in the system catalog and, if necessary, will request the operator to mount the volume that contains it.

2. Confirmation that the data set already exists (DISP parameter).

If the data set is not cataloged, you must, in addition, specify the device that will read the data set and give the serial number of the volume that contains the data set (UNIT and VOLUME parameters).

Parameters of DD Statement		
When required	What you must state	Parameters
Always	Name of data set	DSNAME=
	Disposition of data set	DISP=
If data set not cataloged	Input device	UNIT= or VOLUME=REF=
	Volume serial number	VOLUME=SER=

Figure 11-20. Accessing a REGIONAL Data Set: Essential Parameters of DD Statement

EXAMPLES OF REGIONAL DATA SETS

REGIONAL(1) Data Sets

Figures 11-21 and 11-22 illustrate the creation and updating of a REGIONAL(1) data set.

Figure 11-21 uses the same data as Figure 11-16, but interprets it in a different way: the data set is effectively a list of telephone numbers with the names of the subscribers to whom they are allocated. The telephone numbers correspond with the region numbers in the data set, the data in each occupied region being a subscriber's name. The SPACE parameter of the DD statement requests space for 1000 twenty-byte records (i.e., for 1000 regions); since space is never allocated in units of less than one track and one 2311 track can accommodate 45 twenty-byte records, there will in fact be 1035 regions. Note that there are no recorded keys in a REGIONAL(1) data set.

The data read by the program in Figure 11-22 is identical with that used in Figure 11-17, and the codes are interpreted in the same way. Like Figure 11-17, this program updates the data set and then lists its contents. Note that before each new or updated record is written the existing record in the region is tested to ensure that it is a dummy; this is necessary because a WRITE statement can overwrite an existing record in a REGIONAL(1) data set even if it is not a dummy. Similarly, during the sequential reading and printing of the contents of the data set, each record is tested and dummy records are not printed.

REGIONAL(2) Data Sets

Figures 11-23, 11-24, and 11-25 illustrate the use of REGIONAL(2) data sets. The programs in these figures perform the same functions as those given for REGIONAL(3), with which they can usefully be compared.

The figures depict a library processing scheme, in which loans of books are recorded and reminders are issued for overdue books. Two data sets, STOCK2 and LOANS2 are involved. STOCK2 contains descriptions of the books in the library, and uses the 4-digit book reference numbers as recorded keys; a simple algorithm is used to derive the region numbers from the reference numbers. (It is assumed that there are about 1000 books, each with a number in the range 1000-9999.) LOANS2 contains records of books that are on loan; each record comprises two dates, the date of issue and the date of the last reminder. Each reader is identified by a 3-digit reference number, which is used as a region number in LOANS2; the reader and book numbers are concatenated to form the recorded keys.

In Figure 11-23, the data sets STOCK2 and LOANS2 are created. The file LOANS, which is used to create the data set; LOANS2, is opened for direct output merely to format the data set; the file is closed immediately without any records being written onto the data set. It is assumed that the number of books on loan will not exceed 100; therefore the SPACE parameter in the DD statement that defines LOANS2 requests 100 blocks of 19 bytes (12 bytes of data and a 7-byte key: see Figure 11-24). Direct creation is also used for the data set STOCK2 because, even if the input data is presented in ascending reference number order, identical region numbers might be derived from successive reference numbers.

```
//J010PGEX JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 CRR1:   PROC OPTIONS(MAIN);
         DCL NOS FILE RECORD OUTPUT DIRECT KEYED ENV(REGIONAL(1)),
             CARD CHAR(80),
             NAME CHAR(20) DEF CARD,
             NUMBER CHAR(3) DEF CARD POS(21),
             IOFIELD CHAR(20);
         ON ENDFILE (SYSIN) GO TO FINISH;
         OPEN FILE(NOS);
 NEXT:   GET FILE(SYSIN) EDIT(CARD)(A(80));
         IOFIELD=NAME;
         WRITE FILE(NOS) FROM(IOFIELD) KEYFROM(NUMBER);
         GO TO NEXT;
 FINISH: CLOSE FILE(NOS);
         END CRR1;
/*
//GO.NOS DD UNIT=2311,SPACE=(20,1000),DCB=(RECFM=F,BLKSIZE=20,
//             DSORG=DA),DISP=(NEW,KEEP,DELETE),DSNAME=NOS,
//             VOLUME=SER=D186
//GO.SYSIN DD *
ACTION,G.           162
BAKER,R.            152
BRAMLEY,O.H.        248
CHEESMAN,L.         141
CORY,G.             336
ELLIOTT,D.          875
FIGGINS,E.S.        413
HARVEY,C.D.W.       205
HASTINGS,G.M.       391
KENDALL,J.G.        294
LANCASTER,W.R.      624
MILES,R.            233
NEWMAN,M.W.         450
PITT,W.H.           515
ROLF,D.E.           114
SHEERS,C.D.         241
SUTCLIFFE,M.        472
TAYLOR,G.C.         407
WILTON,L.W.         404
WINSTONE,E.M.       307
/*
```

Figure 11-21. Creating a REGIONAL(1) Data Set

Figure 11-24 illustrates the updating of the data set LOANS2. Each item of input data, read from a punched card, comprises a book number, a reader number, and a code to indicate whether it refers to a new issue (I), a returned book (R), or a renewal (A). The position of the reader number on the card allows the 8-character region number to be derived directly by overlay defining. The DATE built-in function is used to obtain the current date. This date is written in both the issue-date and reminder-date portions of a new record or an updated record.

The program in Figure 11-25 uses a sequential update file (LOANS) to process the records in the data set LOANS2, and a direct input file (STOCK) to obtain the book description from the data set STOCK2 for use in a reminder note. Each record from LOANS2 is tested to see whether the last reminder was issued more than a month ago; if necessary, a reminder note is issued and the current date is written in the reminder-date field of the record.

REGIONAL(3) Data Sets

Figure 11-26, 11-27, and 11-28, which illustrate the use of REGIONAL(3) data sets, are similar to the REGIONAL(2) figures, above; only the important differences are discussed here.

To conserve space in the data set STOCK3, U-format records are used. In each record, the author's name and the title of

```
//J042PGEX JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 ACR1:    PROC OPTIONS(MAIN);
          DCL NOS FILE RECORD KEYED ENV(REGIONAL(1)),
              NAME CHAR(20),
              (NEWNO,OLDNO) CHAR(3),
              CODE CHAR(1),
              IOFIELD CHAR(20),
              BYTE1 CHAR(1) DEF IOFIELD;
          ON ENDFILE(SYSIN) GO TO PRINT;
          OPEN FILE(NOS) DIRECT UPDATE;
 NEXT:    GET FILE(SYSIN) EDIT(NAME,NEWNO,OLDNO,CODE)
                 (A(20),2 A(3),X(53),A(1));
          IF CODE='A' THEN GO TO RITE;
          ELSE IF CODE='C' THEN DO;
            DELETE FILE(NOS) KEY(OLDNO);
            GO TO RITE;
            END;
               ELSE IF CODE='D' THEN DELETE FILE(NOS) KEY(OLDNO);
                    ELSE PUT FILE(SYSPRINT) SKIP EDIT('INVALID CODE:',
                                        NAME)(A(15),A);
          GO TO NEXT;
 RITE:    READ FILE(NOS) KEY(NEWNO) INTO(IOFIELD);
          IF UNSPEC(BYTE1)=(8)'1'B THEN WRITE FILE(NOS) KEYFROM(NEWNO)
            FROM(NAME);
          ELSE PUT FILE(SYSPRINT) SKIP EDIT
                     ('DUPLICATE:',NAME)(A(15),A);
          GO TO NEXT;
 PRINT:   CLOSE FILE(NOS);
          PUT FILE(SYSPRINT) PAGE;
          OPEN FILE(NOS) SEQUENTIAL INPUT;
          ON ENDFILE(NOS) GO TO FINISH;
 NEXTIN:  READ FILE(NOS) INTO(IOFIELD) KEYTO(NEWNO);
          IF UNSPEC(BYTE1)=(8)'1'B THEN GO TO NEXTIN;
          ELSE PUT FILE(SYSPRINT) SKIP EDIT(NEWNO,IOFIELD)(A(5),A);
          GO TO NEXTIN;
 FINISH:  CLOSE FILE(NOS);
          END ACR1;
/*
//GO.NOS DD DSNAME=NOS,DISP=(OLD,KEEP),UNIT=2311,VOLUME=SER=D186
//GO.SYSIN DD *
NEWMAN,M.W.          516450                                          C
GOODFELLOW,D.T.      889                                             A
MILES,R.                233                                          D
HARVEY,C.D.W.        209                                             A
BARTLETT,S.G.        183                                             A
CORY,G.                 336                                          D
READ,K.M.            001                                             A
PITT,W.H.               515
ROLF,D.F.               114                                          D
ELLIOTT,D.           472875                                          C
HASTINGS,G.M.           391                                          D
BRAMLEY,O.H.         439248                                          C
/*
```

Figure 11-22. Accessing a REGIONAL(1) Data Set

the book are concatenated in a single character string, and the lengths of the two parts of the string are written as part of the record. CONTROLLED storage is used for the structure in which the records are built because varying-length strings are not permitted by the PL/I (F) compiler in structures that are referred to in record-oriented transmission statements.

The average record (including the recorded key) is assumed to be 60 bytes; therefore the average number of records per track (i.e., per region) is 25, and there will be 40 regions.

In Figure 11-26, the data set STOCK3 is created sequentially; duplicate region

```
//J011PGEX JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 CRR2:   PROC OPTIONS(MAIN);
         DCL (LOANS,STOCK) FILE RECORD KEYED ENV(REGIONAL(2)),
             NUMBER CHAR(4),
             1 BOOK,
               2 AUTHOR CHAR(25),
               2 TITLE CHAR(50),
               2 QTY FIXED DEC(3),
             INTER FIXED DEC(5),
             REGION CHAR(8);
         ON ENDFILE(SYSIN) GO TO FINISH;
         OPEN FILE(LOANS) DIRECT OUTPUT;
         CLOSE FILE(LOANS);
         OPEN FILE(STOCK) DIRECT OUTPUT;
 NEXT:   GET FILE(SYSIN) LIST(NUMBER,BOOK);
         INTER=(NUMBER-1000)/9;
         REGION=INTER;
         WRITE FILE(STOCK) FROM(BOOK) KEYFROM(NUMBER||REGION);
         GO TO NEXT;
 FINISH: CLOSE FILE(STOCK);
         END CRR2;
/*
//GO.LOANS DD UNIT=2311,SPACE=(19,100),DCB=(RECFM=F,BLKSIZE=12,
//             DSORG=DA,KEYLEN=7),DISP=(NEW,KEEP,DELETE),DSNAME=LOANS2,
//             VOLUME=SER=D186
//GO.STOCK DD UNIT=2311,SPACE=(81,(100,20)),DCB=(RECFM=F,BLKSIZE=77,
//             DSORG=DA,KEYLEN=4),DISP=(NEW,KEEP,DELETE),DSNAME=STOCK2,
//             VOLUME=SER=D186
//GO.SYSIN DD *
'1015' 'W.H.AINSWORTH' 'THE ADMIRABLE CRICHTON'  1
'1214' 'L.CARROLL' 'THE HUNTING OF THE SNARK'  1
'3079' 'G.FLAUBERT' 'MADAME BOVARY'  1
'3083' 'V.M.HUGO' 'LES MISERABLES'  2
'3085' 'J.K.JEROME' 'THREE MEN IN A BOAT'  2
'4295' 'W.LANGLAND' 'THE BOOK CONCERNING PIERS THE PLOWMAN'  1
'5998' 'W.SHAKESPEARE' 'MUCH ADO ABOUT NOTHING' 3
'6591' 'F.RABELAIS' 'THE HEROIC DEEDS OF GARGANTUA AND PANTAGRUEL'  1
'8362' 'H.D.THOREAU' 'WALDEN, OR LIFE IN THE WOODS'  1
'9765' 'H.G.WELLS' 'THE TIME MACHINE'  3
/*
```

Figure 11-23. Creating a REGIONAL(2) Data Set

numbers are acceptable since each region can contain more than one record.

In Figure 11-27, the region number for the data set LOANS3 is obtained simply by testing the reader number; there are only three regions, since a 2311 track can hold 36 nineteen-byte records.

The only notable difference between Figure 11-28 and the corresponding REGIONAL(2) figure is in the additonal processing required for the analysis of the records read from the data set STOCK3. The records are read into a varying-length character string and a based structure is overlaid on the string so that the data in the record can be extracted.

```
//J043PGE1 JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=99999',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 DUR2:    PROC OPTIONS(MAIN);
          DCL 1 RECORD,2(ISSUE,REMINDER) CHAR(6),
              SYSIN FILE RECORD INPUT SEQUENTIAL,
              LOANS FILE RECORD UPDATE DIRECT KEYED ENV(REGIONAL(2)),
              CARD CHAR(80),
              BOOK CHAR(4) DEF CARD,
              READER CHAR(3) DEF CARD POS(10),
              CODE CHAR(1) DEF CARD POS(20),
              REGION CHAR(8) DEF CARD POS(5);
          ON ENDFILE(SYSIN) GO TO FINISH;
          OPEN FILE(SYSIN),FILE(LOANS);
          ISSUE,REMINDER=DATE;
 NEXT:    READ FILE(SYSIN) INTO(CARD);
          IF CODE='I' THEN WRITE FILE(LOANS) FROM(RECORD)
                           KEYFROM(READER||BOOK||REGION);
          ELSE IF CODE='R' THEN DELETE FILE(LOANS)
                           KEY(READER||BOOK||REGION);
          ELSE IF CODE='A' THEN REWRITE FILE(LOANS) FROM(RECORD)
                           KEY(READER||BOOK||REGION);
          ELSE PUT FILE(SYSPRINT) SKIP LIST
                           ('INVALID CODE:',BOOK,READER);
          GO TO NEXT;
 FINISH:  CLOSE FILE(SYSIN),FILE(LOANS);
          END DUR2;
/*
//GO.LOANS DD DSNAME=LOANS2,DISP=(OLD,KEEP),UNIT=2311,VOLUME=SER=D186
//GO.SYSIN DD *
3517      095        X
5999      003        A
3083      091        R
1214      049        I
/*
```

Figure 11-24. REGIONAL(2) Data Sets: Direct Update

```
//J043PGE3 JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 SUR2:    PROC OPTIONS(MAIN);
          DCL LOANS FILE RECORD SEQUENTIAL UPDATE KEYED ENV(REGIONAL(2)),
              STOCK FILE RECORD DIRECT INPUT KEYED ENV(REGIONAL(2)),
              (TODAY,LASMTH) CHAR(6),
              YEAR PIC '99' DEF LASMTH,
              MONTH PIC '99' DEF LASMTH POS(3),
              1 RECORD,2(ISSUE,REMINDER) CHAR(6),
              LOANKEY CHAR(7),
              READER CHAR(3) DEF LOANKEY,
              BKNO CHAR(4) DEF LOANKEY POS(4),
              INTER FIXED DEC(5),
              REGION CHAR(8),
              1 BOOK,
                2 AUTHOR CHAR(25),
                2 TITLE CHAR(50),
                2 QTY FIXED DEC(3);
          TODAY,LASMTH=DATE;
          IF MONTH='01' THEN DO;
            MONTH='12';
            YEAR=YEAR-1;
            END;
          ELSE MONTH=MONTH-1;
          OPEN FILE(LOANS),FILE(STOCK);
          ON ENDFILE(LOANS) GO TO FINISH;
 NEXT:    READ FILE(LOANS) INTO(RECORD) KEYTO(LOANKEY);
          IF REMINDER<LASMTH THEN DO;
            REMINDER=TODAY;
            REWRITE FILE(LOANS) FROM(RECORD);
            INTER=(BKNO-1000)/9;
            REGION=INTER;
            READ FILE(STOCK) INTO(BOOK) KEY(BKNO||REGION);
            PUT FILE(SYSPRINT) SKIP(4) EDIT(READER,AUTHOR,TITLE)
            (A,SKIP(2));
            END;
          GO TO NEXT;
 FINISH:  CLOSE FILE(LOANS),FILE(STOCK);
          END SUR2;
/*
//GO.LOANS DD DSNAME=LOANS2,DISP=(OLD,KEEP),UNIT=2311,VOLUME=SER=D186
//GO.STOCK DD DSNAME=STOCK2,DISP=(OLD,KEEP),UNIT=2311,VOLUME=SER=D186
```

Figure 11-25. REGIONAL(2) Data Sets: Sequential Update and Direct Input

```
//J006PGEX JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 CRR3:   PROC OPTIONS(MAIN);
         DCL (LOANS,STOCK) FILE RECORD KEYED ENV(REGIONAL(3)),
             1 CARD,
               2 NUMBER CHAR(4),
               2 AUTHOR CHAR(25) VAR,
               2 TITLE CHAR(50) VAR,
               2 QTY1 FIXED DEC(3),
             (L1,L2,X) FIXED DEC(3),
             1 BOOK CTL,
               2 (L3,L4) FIXED DEC(3),
               2 QTY2 FIXED DEC(3),
               2 DESCN CHAR(X),
             INTER FIXED DEC(5),
             REGION CHAR(8);
         ON ENDFILE(SYSIN) GO TO FINISH;
         OPEN FILE(LOANS) DIRECT OUTPUT;
         CLOSE FILE(LOANS);
         OPEN FILE(STOCK) SEQUENTIAL OUTPUT;
 NEXT:   GET FILE(SYSIN) LIST(CARD);
         L1=LENGTH(AUTHOR);
         L2=LENGTH(TITLE);
         X=L1+L2;
         ALLOCATE BOOK;
         L3=L1;
         L4=L2;
         QTY2=QTY1;
         DESCN=AUTHOR||TITLE;
         INTER=(NUMBER-1000)/225;
         REGION=INTER;
         WRITE FILE(STOCK) FROM(BOOK) KEYFROM(NUMBER||REGION);
         FREE BOOK;
         GO TO NEXT;
 FINISH: CLOSE FILE(STOCK);
         END CRR3;
/*
//GO.LOANS DD UNIT=2311,SPACE=(19,100),DCB=(RECFM=F,BLKSIZE=12,
//             DSORG=DA,KEYLEN=7),DISP=(NEW,KEEP,DELETE),DSNAME=LOANS3,
//             VOLUME=SER=D186
//GO.STOCK DD UNIT=2311,SPACE=(60,(1000,20)),DCB=(RECFM=U,BLKSIZE=110,
//             DSORG=DA,KEYLEN=4),DISP=(NEW,KEEP,DELETE),DSNAME=STOCK3,
//             VOLUME=SER=D186
//GO.SYSIN DD *
'1015' 'W.H.AINSWORTH' 'THE ADMIRABLE CRICHTON'  1
'1214' 'L.CARROLL' 'THE HUNTING OF THE SNARK'  1
'3079' 'G.FLAUBERT' 'MADAME BOVARY'  1
'3083' 'V.M.HUGO' 'LES MISERABLES'  2
'3085' 'J.K.JEROME' 'THREE MEN IN A BOAT'  2
'4295' 'W.LANGLAND' 'THE BOOK CONCERNING PIERS THE PLOWMAN'  1
'5998' 'W.SHAKESPEARE' 'MUCH ADO ABOUT NOTHING' 3
'6591' 'F.RABELAIS' 'THE HEROIC DEEDS OF GARGANTUA AND PANTAGRUEL'  1
'8362' 'H.D.THOREAU' 'WALDEN, OR LIFE IN THE WOODS'  1
'9765' 'H.G.WELLS' 'THE TIME MACHINE'  3
/*
```

Figure 11-26. Creating a REGIONAL(3) Data Set

```
//J044PGE1 JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 DUR3:    PROC OPTIONS(MAIN);
          DCL 1 RECORD,2(ISSUE,REMINDER) CHAR(6),
              SYSIN FILE RECORD INPUT SEQUENTIAL,
              LOANS FILE RECORD UPDATE DIRECT KEYED ENV(REGIONAL(3)),
              CARD CHAR(80),
              BOOK CHAR(4) DEF CARD,
              READER CHAR(3) DEF CARD POS(10),
              CODE CHAR(1) DEF CARD POS(20),
              REGION CHAR(8);
          ON ENDFILE(SYSIN) GO TO FINISH;
          OPEN FILE(SYSIN),FILE(LOANS);
          ISSUE,REMINDER=DATE;
 NEXT:    READ FILE(SYSIN) INTO(CARD);
          IF READER<'034' THEN REGION='00000000';
          ELSE IF READER<'067' THEN REGION='00000001';
               ELSE REGION='00000002';
          IF CODE='I' THEN WRITE FILE(LOANS) FROM(RECORD)
                                KEYFROM(READER||BOOK||REGION);
          ELSE IF CODE='R' THEN DELETE FILE(LOANS)
                                KEY(READER||BOOK||REGION);
               ELSE IF CODE='A' THEN REWRITE FILE(LOANS) FROM(RECORD)
                                KEY(READER||BOOK||REGION);
                    ELSE PUT FILE(SYSPRINT) SKIP LIST
                                ('INVALID CODE:',BOOK,READER);
          GO TO NEXT;
 FINISH:  CLOSE FILE(SYSIN),FILE(LOANS);
          END DUR3;
/*
//GO.LOANS DD DSNAME=LOANS3,DISP=(OLD,KEEP),UNIT=2311,VOLUME=SER=D186
//GO.SYSIN DD *
3517      095       X
5999      003       A
3083      091       R
1214      049       I
/*
```

Figure 11-27. REGIONAL(3) Data Sets: Direct Update

```
//J044PGE3 JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED='LIST'
//PL1L.SYSIN DD *
 SUR3:   PROC OPTIONS(MAIN);
         DCL LOANS FILE RECORD SEQUENTIAL UPDATE KEYED ENV(REGIONAL(3)),
             STOCK FILE RECORDS DIRECT INPUT KEYED ENV(REGIONAL(3)),
             (TODAY,LASMTH) CHAR(6),
             YEAR PIC '99' DEF LASMTH,
             MONTH PIC '99' DEF LASMTH POS(3),
             1 RECORD,2(ISSUE,REMINDER) CHAR(6),
             LOANKEY CHAR(7),
             READER CHAR(3) DEF LOANKEY,
             BKNO CHAR(4) DEF LOANKEY POS(4),
             INTER FIXED DEC(5),
             REGION CHAR(8),
             INREC CHAR(81) VAR,
             1 BOOK BASED(P),
               2 (L1,L2) FIXED DEC(3),
               2 QTY FIXED DEC(3),
               2 DESCN CHAR(75),
             AUTHOR CHAR(25) VAR,
             TITLE CHAR(50) VAR;
         TODAY,LASMTH=DATE;
         IF MONTH='01' THEN DO;
           MONTH='12';
           YEAR=YEAR-1;
           END;
         ELSE MONTH=MONTH-1;
         OPEN FILE(LOANS),FILE(STOCK);
         ON ENDFILE(LOANS) GO TO FINISH;
 NEXT:   READ FILE(LOANS) INTO(RECORD) KEYTO(LOANKEY);
         IF REMINDER<LASMTH THEN DO;
           REMINDER=TODAY;
           REWRITE FILE(LOANS) FROM(RECORD);
         INTER=(BKNO-1000)/225;
           REGION=INTER;
         READ FILE(STOCK) INTO(INREC) KEY(BKNO||REGION);
                   DCL STR BIT(648);
                   STR=UNSPEC(INREC);
                   PUT FILE(SYSPRINT) PAGE DATA(STR);
         P=ADDR(INREC);
         AUTHOR=SUBSTR(DESCN,1,L1);
         TITLE=SUBSTR(DESCN,L1+1,L2);
           PUT FILE(SYSPRINT) SKIP(4) EDIT(READER,AUTHOR,TITLE)
           (A,SKIP(2));
           END;
         GO TO NEXT;
 FINISH: CLOSE FILE(LOANS),FILE(STOCK);
         END SUR3;
/*
//GO.LOANS DD DSNAME=LOANS3,DISP=(OLD,KEEP),UNIT=2311,VOLUME=SER=D186
//GO.STOCK DD DSNAME=STOCK3,DISP=(OLD,KEEP),UNIT=2311,VOLUME=SER=D186
```

Figure 11-28. REGIONAL(3) Data Sets: Sequential Update and Direct Input

Teleprocessing

INTRODUCTION

The teleprocessing facilities of PL/I are provided by an extension of the basic record-oriented transmission facilities with the addition of the TRANSIENT file attribute and of the PENDING condition. The (F) compiler provides a communication link between PL/I message processing programs (MPPs) using these features and the Queued Telecommunications Access Method (QTAM) of the operating system.

A QTAM message control program (MCP) handles messages originating from and destined for a number of remote terminals, each of which is identified by a terminal name carried with the message. These messages are transmitted to and from your PL/I message processing program via queues

in main storage. (These queues are supported by corresponding intermediate queues in auxiliary storage on a disk data set. Your PL/I program has access only to the main-storage queues by means of a single intermediate buffer for each file.)

The exact message format (specified to the compiler by means of the ENVIRONMENT attribute) depends on the MCP, but each message will carry the terminal name with it. A message may be a complete unit, or may consist of a number of records so that it can be split up for processing. You must have this message format information to enable you to write the message processing program. Full information on how to write an MPP is given in IBM System/360 Operating System: PL/I (F) Language Reference Manual. A full account of QTAM procedure is given in: IBM System/360 Operating System: QTAM Message Processing Program Services and IBM System/360 Operating System: QTAM Message Control Program.

MESSAGE PROCESSING PROGRAM (MPP)

This program receives the terminal message as input and produces output according to the data in the message. You can code this program in PL/I.

An MPP is not mandatory at a teleprocessing installation, as for example, an MCP is. If the messages you transmit do not require processing, because they are only switched between terminals, an MPP is not required. However, you can pass data to a problem program and you can receive the output with a minimum of delay, and most installations are likely to have a set of processing programs available for these purposes. These programs are stored as load modules, either in main storage or in a library in auxiliary storage.

HOW TO RUN AN MPP

Figure 11-29 shows an example of an MPP and the job control language required to run it. The EXEC statement invokes the cataloged procedure PL1LFCL to compile and link-edit the PL/I message processing program. The load module produced is stored in the partitioned data set SYS1.MSGLIB under the member name MPPROC. The telecommunications library, SYS1.TELCMLIB, is concatenated with the PL/I library, SYS1.PL1LIB.

In the PL/I message processing program, MPP is declared as a teleprocessing file that can process messages up to 100 bytes

```
//           JOB
//           EXEC PL1LFCL
//PL1L.SYSIN DD *
  MPPROC: PROC OPTIONS(MAIN);
          DCL MPP FILE RECORD KEYED TRANSIENT ENV(G(100)),
              OUTMSG FILE RECORD KEYED TRANSIENT ENV(G(500)),
              INDATA CHAR(100),
              OUTDATA CHAR(500),
              TKEY CHAR(6);
                .
                .
                .
          OPEN FILE(MPP) INPUT,FILE(OUTMSG) OUTPUT;
                .
                .
                .
          READ FILE(MPP) KEYTO(TKEY) INTO(INDATA);
                .
                .
                .
          WRITE FILE(OUTMSG) KEYFROM(TKEY) FROM(OUTDATA);
                .
                .
                .
  ENDTP:  CLOSE FILE(MPP),FILE(OUTMSG);
          END MPPROC;
/*
//LKED.SYSLIB DD DSNAME=SYS1.PL1LIB,DISP=SHR
//           DD DSNAME=SYS1.TELCMLIB,DISP=SHR
//LKED.SYSLMOD DD DSNAME=SYS1.MSGLIB(MPPROC),...
```

Figure 11-29. PL/I Message Processing Program

long. Similarly OUTMSG is declared as a teleprocessing file that can receive messages up to 500 bytes long.

The READ statement gets a record (message or message segment) from the queue. The terminal identifier is inserted into the KEYTO character string. The record is placed in the INDATA variable for processing. The appropriate READ SET statement could also have been used here.

The WRITE statement puts the data in OUTDATA into the destination queue; the terminal identifier is taken from the character string in TKEY. An appropriate LOCATE statement could also have been used.

Once the load module has been stored in auxiliary storage on a direct-access device it can be restored for execution at any time. The job control statements to perform this might be:

```
//          JOB
//JOBLIB    DD   DSNAME=SYS1.MSGLIB,DISP=SHR
//          EXEC PGM=MPPROC
//MPP       DD   DUMMY
//OUTMSG    DD   DUMMY
//SYSPRINT  DD   SYSOUT=A
```

The JOBLIB DD statement is required to make SYS1.MSGLIB available for resolution of external references. The DD statements for MPPROC associate the PL/I files MPP and OUTMSG with their respective main-storage process queues. As no input/output operations are required these are DD DUMMY statements.

Chapter 12: Libraries of Data Sets

Introduction

Within IBM System/360 Operating System, the terms 'library' and 'partitioned data set' are used synonymously to signify a type of data set that can be used for the storage of other (sequential) data sets (usually programs in the form of source, object, or load modules). 'Partitioned data set' is the more precise term, but its use is generally confined to contexts where the physical structure of the data set is more significant than its conceptual nature; otherwise, 'library' is the term in general use.

There are three types of library:

1. A system library is a partitioned data set that houses frequently used programs; system libraries form an integral part of the operating system. The most important system libraries for the PL/I programmer are the link library (named SYS1.LINKLIB), the cataloged-procedure library (SYS1.PROCLIB), and the PL/I subroutine library (SYS1.PL1LIB).

2. A temporary library is a partitioned data set that exists only for the duration of a job. This type of library is particularly useful for containing the output from a linkage editor run until it is executed by a later job step.

3. A private library is any permanent partitioned data set that is not part of the operating system. Private libraries are often used to house groups of programs not used frequently enough to warrant their inclusion in a system library. They are usually made available to a job by a special DD statement with the ddname JOBLIB; they can also be accessed by being concatenated with a system library and retrieved by use of the system library ddname.

Structure of a Partitioned Data Set

A partitioned data set, which must be on a direct-access device and wholly contained on one volume, contains independent sequentially organized (CONSECUTIVE) data sets, each termed a member. Each member has a unique name not more than eight characters long stored in a directory that is part of the data set. The directory permits direct access to any member; the members themselves are always processed sequentially.

Individual members can be added to or deleted from a partitioned data set as required. When a member is deleted, the member name is removed from the directory, but the space occupied by the member cannot be reused. If there is not sufficient space available in the directory for an additional entry, or not enough space available within the data set for an additional member, no new members can be stored.

DIRECTORY

The directory of a partitioned data set is a series of records at the beginning of the data set; there is at least one record (directory entry) for each member of the data set. Each directory entry contains a member name, the relative address of the member within the data set, and a variable amount of user data. The entries are arranged in ascending alphameric order of member names (Figure 12-1).

Although directory entries do not have a fixed length, they are blocked into fixed-length blocks of 256 bytes. Each block contains a 2-byte count field and as many complete entries as will fit into the remaining 254 bytes. The count field specifies the number of active bytes in the block. Figure 12-2 illustrates the format of a directory block. The directory is in effect a sequential data set that contains fixed-length unblocked records, and can be read as such.

Figure 12-5 demonstrates a method of extracting information from directory entries. (The program lists the names of all the members of any library; the library must be named at execution time in a DD statement with the ddname LINK.)

The member name, which may be an alias, occupies the first eight bytes of a directory entry, and the relative address of the member within the data set the next three bytes (Figure 12-3). The twelfth byte contains the following information:

Bit	Description
0	If this bit is set to 1, the name in the member-name field is an alias.
1,2	Binary value that indicates the number of pointers in the user data field. These pointers contain the relative addresses of locations within the member. You cannot use a PL/I program to write such pointers; they can be inserted only by means of the assembler language STOW macro instruction.
3-7	Binary value that indicates the number of halfwords in the user data field (including pointers).

A directory entry can contain up to 62 bytes of user data (information inserted by the program that created the member). An entry that refers to a member (load module) written by the linkage editor includes user data in a standard format, which is described in IBM System/360 Operating System: System Control Blocks. If you use a PL/I program to create a member, the operating system creates the directory entry for you and you cannot write any user data. However, you can use assembler language macro instructions to create a member and write your own user data; the method is described in IBM System/360 Operating System: Supervisor and Data Management Services.

Creating a Partitioned Data Set

The simplest way to create a partitioned data set is to include in any job step of any job a DD statement that contains the following information:

	Parameter of DD Statement[1]
Type of device that will process the data set	UNIT=
Serial number of the volume that will contain the data set	VOLUME=SER=
Name of the data set	DSNAME=
Amount of space required for the data set	SPACE=
Disposition of the data set	DISP=

[1]Appendix B describes all the parameters of the DD statement referred to in this chapter.

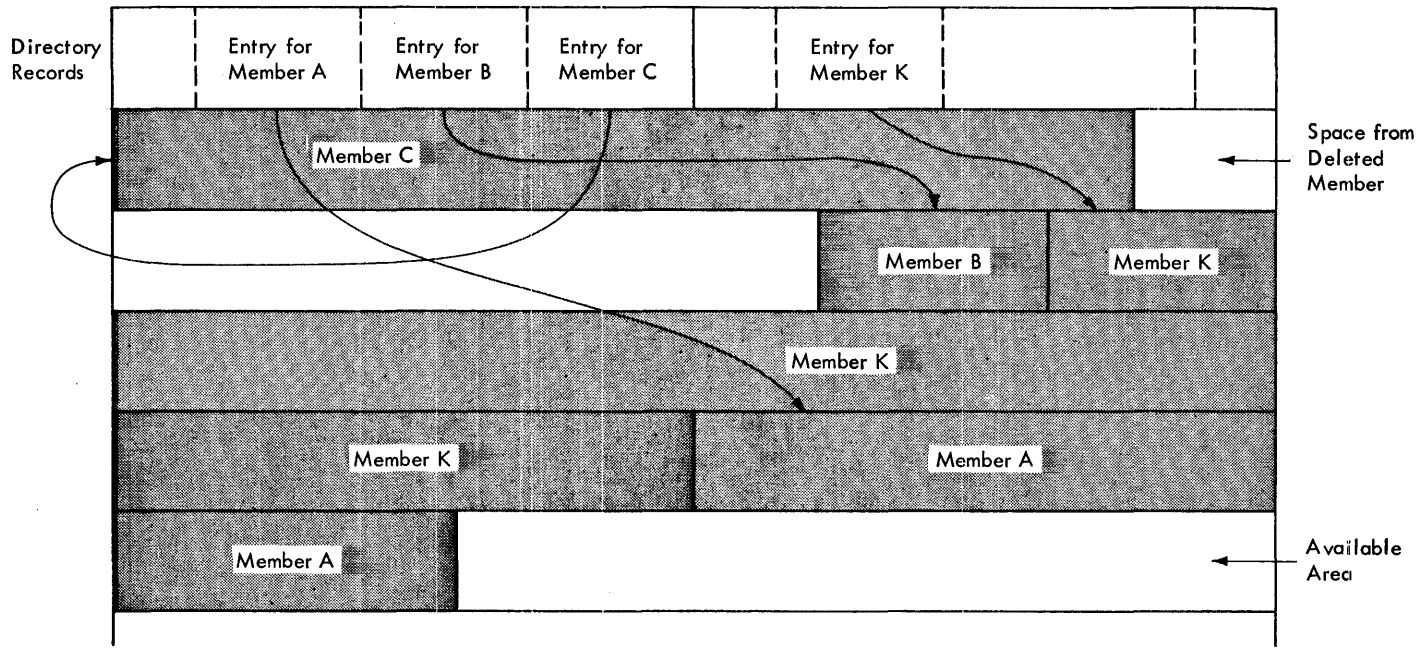


Figure 12-1. A Partitioned Data Set

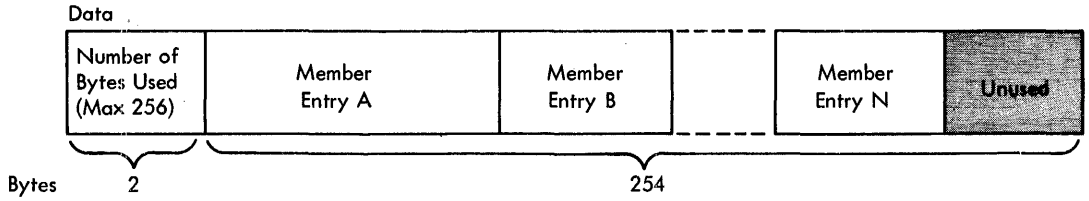
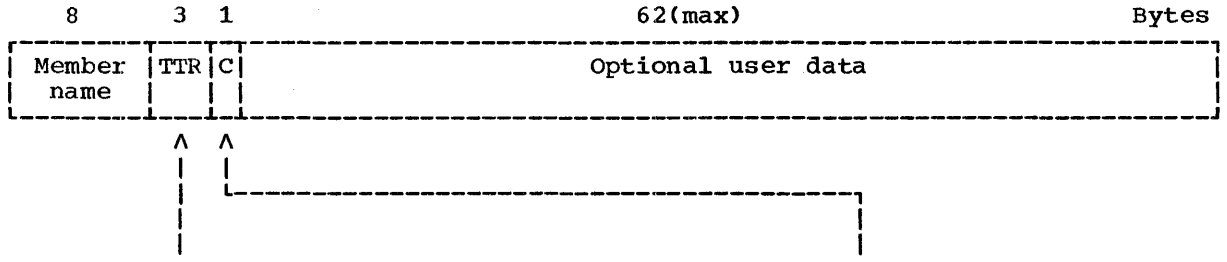


Figure 12-2. A Partitioned Data Set Directory Block



Pointer to start of member:

TT - track number relative to start of data set

R - relative block number on track

Bit	
0	If set to 1, name is an alias
1,2	Number of pointers in user data field
3-7	Binary value indicating number of halfwords of user data (including pointers)

Figure 12-3. Contents of Directory Entry

For example, the DD statement:

```
//PDS DD UNIT=2311,VOLUME=SER=3412,
//             DSNAME=ALIB,
//             SPACE=(CYL,(50,,10)),
//             DISP=(,CATLG)
```

requests the job scheduler to allocate 50 cylinders of the 2311 disk pack with serial number 3412 for a new partitioned data set named ALIB, and to enter this name in the system catalog. The last term of the SPACE parameter requests that part of the space allocated to the data set be reserved for ten directory blocks; it is the presence of this term that indicates to the job scheduler that the request is for a partitioned data set[1].

If you want to insert the first member in the data set at the time you create it, you must include the DD statement in the job step that writes the member, and the DSNAME parameter must include the member name in parentheses. For example, DSNAME=ALIB(MEM1) names the member MEM1 in the data set ALIB. If the member is placed in the data set by the linkage editor, you can use the linkage-editor NAME statement or the compiler OBJNM option instead of including the member name in the DSNAME parameter.

[1]The SPACE parameter in a DD statement that defines an INDEXED data set can include a term in this position to indicate the size of the index, but the DD statement must also include the DCB subparameter DSORG=IS.

SPACE PARAMETER

The SPACE parameter in a DD statement that defines a new partitioned data set must always be of the form SPACE=(units, (quantity,increment,directory)). Although you can omit the third term (increment), indicating its absence by a comma, the last term, which specifies the number of directory blocks to be allocated, must always be present.

The amount of space required for a partitioned data set depends on the number and sizes of the members to be stored in it and on how often members will be added or replaced. (Space occupied by deleted members is not released.) The number of directory blocks required depends on the

number of members and the number of aliases. Although you can specify an incremental quantity in the SPACE parameter that will allow the operating system to obtain more space for the data set if necessary, both at the time of creation and when new members are added, the number of directory blocks is fixed at the time of creation and cannot be increased.

If the data set is likely to be large or you expect to do a lot of updating, it might be best to allocate a full volume. Otherwise, make your estimate as accurate as possible to avoid wasting space or time recreating the data set.

The number of entries that a 256-byte directory block can contain depends on the amount of user data included in the entries. The maximum length of an entry is 74 bytes, but the entries produced by the linkage editor vary in length between 34 bytes and 52 bytes, which is equivalent to between four and seven entries per block.

Processing a Member

When programming in PL/I, you will ordinarily use partitioned data sets only for storing source, object, or load modules, and the modules themselves will be created by the compiler or the linkage editor. You need only include an appropriately named DD statement to indicate the names of the partitioned data set and the new member. (Note that you should restrict the use of a particular library to the storage of one type of module, since all members of a partitioned data set must have identical characteristics.)

Figure 12-4 illustrates the use of the cataloged procedure PL1LFC to compile a simple PL/I program and place the object module in a new library named EXLIB. The DD statement that defines the new library and names the object module overrides the DD statement SYSLIN in the cataloged procedure. (The PL/I program is a function procedure that, given two values in the form of the character string produced by the TIME built-in function, returns the difference in milliseconds.)

Figure 12-5 illustrates the use of the cataloged procedure PL1LFCL to compile and link-edit a PL/I program and place the load module in the existing library 'FLM'. (The PL/I program lists the names of the members of a library.)

You can delete a member of a partitioned data set by means of the SCRATCH statement of the operating system utility program IEHPROGM, which is described in IBM System/360 Operating Sytem: Utilitites. The SCRATCH statement deletes only the directory entry that refers to the member; you cannot free the space occupied by the member unless you reorganize the entire data set. Do not attempt to delete a member by including the parameter DISP=(OLD,DELETE) in the DD statement that defines it; this would result in the deletion of the entire data set.

To reorganize a partitioned data set, you must copy the members into a temporary partitioned data set, delete and recreate the original data set, and copy the members back into it. The system utility programs include facilities for copying members of partitioned data sets.

```
//J057PGEX JOB
//CO EXEC PL1LFC,PARM.PL1L='SIZE=999999'
//PL1L.SYSLIN DD UNIT=2311,VOLUME=SER=D186,DSNAME=EXLIB(ELAPSE),
//             SPACE=(CYL,(10,,2)),DISP=(NEW,KEEP)
//PL1L.SYSIN DD *
 ELAPSE: PROC(TIME1,TIME2);
         DCL (TIME1,TIME2) CHAR(9),
             H1 PIC '99' DEF TIME1,
             M1 PIC '99' DEF TIME1 POS(3),
             MS1 PIC '99999' DEF TIME1 POS(5),
             H2 PIC '99' DEF TIME2,
             M2 PIC '99' DEF TIME2 POS(3),
             MS2 PIC '99999' DEF TIME2 POS(5),
             ETIME FIXED DEC(7);
         IF H2<H1 THEN H2=H2+24;
         ETIME=((H2*60+M2)*60000+MS2)-((H1*60+M1)*60000+MS1);
         RETURN(ETIME);
         END ELAPSE;
/*
```

Figure 12-4. Placing an Object Module in a New Library

```
//J062PGE2 JOB
//COLE EXEC PL1LFCL,PARM.PL1L='SIZE=999999',PARM.LKED=''
//PL1L.SYSIN DD *
 MNAME:    PROC OPTIONS(MAIN);
           DCL LINK FILE RECORD SEQUENTIAL INPUT,
              1 DIRBLK,
                2 COUNT BIT(16),
                2 ENTRIES(254) CHAR(1),
              1 ENTRY BASED(A),
                2 NAME CHAR(8),
                2 TTR CHAR(3),
                2 INDIC,
                  3 ALIAS BIT(1),
                  3 TTRS BIT(2),
                  3 USERCT BIT(5),
             (LEN,PTR) FIXED BIN(31);
           ON ENDFILE(LINK) GO TO FINISH;
           OPEN FILE(LINK);
 NEXTBLK:  READ FILE(LINK) INTO(DIRBLK);
           LEN=COUNT;
           PTR=1;
 NEXTENT:  A=ADDR(ENTRIES(PTR));
           PUT FILE(SYSPRINT) SKIP LIST(NAME);
           PTR=PTR+12+2*USERCT;
           IF PTR+2>LEN THEN GO TO NEXTBLK;
           GO  TO NEXTENT;
 FINISH:   CLOSE FILE(LINK);
           END MNAME;
/*
//LKED.SYSLMOD DD UNIT=2311,VOLUME=SER=D186,DSNAME=FLM(DIRLIST),
//             DISP=OLD
```

Figure 12-5. Placing a Load Module in an Existing Library

PROCESSING WITH PL/I

You can use a PL/I program to write a member into a partitioned data set, or to read or update a member. In each case, you must identify the member in an appropriate DD statement and process it as a CONSECUTIVE data set.

Creating a Member

The members of a partitioned data set must have identical characteristics. This is necessary because the volume table of contents (VTOC) will contain only one data set control block (DSCB) for the data set, and not one for each member. Although it is possible to place members with different characteristics into a partitioned data set, you may subsequently have difficulty in retrieving them. Note that, when you use a PL/I program to create a member, the operating system creates the directory entry; you cannot place information in the user data field.

When you create a member in a new partitioned data set, the DD statement that defines the data set must include all the parameters listed under the heading 'Creating a Partitioned Data Set,' above (although you can omit the DISP parameter if the data set is temporary). You must also describe the characteristics of the member (record format, etc.) either in the DCB parameter or in your PL/I program; these characteristics will also apply to other members added to the data set.

However, if the partitioned data set already exists, you will not need the SPACE parameter; the original space allocation applies to the data set and not to an individual member. Furthermore, you will not need to describe the characteristics of the member, since these are already recorded in the DSCB for the data set.

If you want to add two or more members to a partitioned data set in one job step, you must include a DD statement for each member, and you must close one file that refers to the data set before you open another.

Figure 12-6 illustrates the use of a PL/I program to create a CONSECUTIVE data set and place it in a new library.

```
//J059PGEX JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED=''
//PL1L.SYSIN DD *
 NMEM:   PROC OPTIONS(MAIN);
         DCL IN FILE RECORD SEQUENTIAL INPUT,
             OUT FILE RECORD SEQUENTIAL OUTPUT,
             IOFIELD CHAR(80) BASED(A);
         OPEN FILE(IN),FILE(OUT);
         ON ENDFILE(IN) GO TO FINISH;
 NEXT:   READ FILE(IN) SET(A);
         WRITE FILE(OUT) FROM(IOFIELD);
         GO TO  NEXT;
 FINISH: CLOSE FILE(IN),FILE(OUT);
         END NMEM;
/*
//GO.OUT DD UNIT=2311,VOLUME=SER=D186,DSNAME=ALIB(NMEM),
//             DISP=(,KEEP),SPACE=(CYL,(10,1,1)),
//             DCB=(RECFM=FB,BLKSIZE=3600,LRECL=80)
//GO.IN DD *

Insert here data to be placed in member

/*
```

• Figure 12-6. Using a PL/I Program to Create a Member of a Partitioned Data Set

```
//J060PGEX JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='SIZE=999999',PARM.LKED=''
//PL1L.SYSIN DD *
 UPDTM:  PROC OPTIONS(MAIN);
         DCL (OLD,NEW) FILE RECORD SEQUENTIAL,
             DATA CHAR(80);
         ON ENDFILE(OLD) GO TO FINISH;
         OPEN FILE(OLD) INPUT,FILE(NEW) OUTPUT TITLE('OLD');
 NEXT:   READ FILE(OLD) INTO(DATA);
         IF DATA=' ' THEN GO TO NEXT;
         WRITE FILE(NEW) FROM(DATA);
         GO TO NEXT;
 FINISH: CLOSE FILE(OLD),FILE(NEW);
         END UPDTM;
/*
//GO.OLD DD UNIT=2311,VOLUME=SER=D186,DSNAME=ALIB(NMEM),DISP=OLD
```

Figure 12-7. Updating a Member of a Partitioned Data Set

Updating a Member

To use a PL/I program to update one or more records within a member of a partitioned data set, you must rewrite the entire member in another part of the data set; this is rarely an economic proposition, since the space originally occupied by the member cannot be used again. You must use two files in your PL/I program, but both can be associated with the same DD statement. Figure 12-7 is a program that updates the member created in Figure 12-6; it copies all the records of the original member except those that contain only blanks.

Operating System Utility Programs

The operating system includes several utility programs that are useful for processing partitioned data sets. IBM System/360 Operating System: Utilities describes all these utility programs and explains how to use them. The facilities offered by the utility programs include moving, copying, and merging the contents of partitioned data sets, updating members in place, and listing the contents of directories.

System Libraries

LINK LIBRARY

The link library (SYS1.LINKLIB) is a system library that houses frequently used programs. Any program that you name in the PGM parameter of an EXEC statement must be in the link library unless you specify a private library (see below). The link library is always available to all job steps of all jobs. The control program provides the necessary data control block and establishes the logical relationship between your program and the library, making the members of the library available to your program.

An assembler language program can use the macro instructions LINK, XCTL, ATTACH, and LOAD to request the control program to load a program from the link library (or a private library) into main storage and (for LINK, ATTACH, and XCTL only) pass control to it. Although there is no equivalent facility in PL/I (F), some of the PL/I library subroutines use these macro instructions to call other library subroutines.

PROCEDURE LIBRARY

The procedure library (SYS1.PROCLIB) is a system library that houses cataloged procedures. It is normally accessed only by the control program or by the utility program IEBUPDTE, which may be used for adding or changing cataloged procedures. Chapter 8 discusses cataloged procedures and the procedure library.

PL/I SUBROUTINE LIBRARY

The PL/I subroutine library (SYS1.PL1LIB) is a system library that houses a set of load modules that, during execution of a PL/I program, supplement the machine instructions generated by the compiler. These modules can be divided into two groups:

1. Modules that serve as an interface between compiled code and the facilities of the operating system. These modules are concerned primarily with input and output, storage management, and error and interrupt handling.
2. Modules that perform data processing operations during program execution. These modules handle, for example, input/output editing, data conversion, and many of the PL/I built-in functions.

Certain modules are loaded dynamically during the execution of a program. These modules reside in the link library (SYS1.LINKLIB): they are transient modules and are loaded, when required, by the macro instructions LINK, LOAD and XCTL. The link library modules comprise:

1. The print and message modules of the error and interrupt-handling subroutines.
2. The modules for opening and closing files.
3. The record-oriented transmission modules.

Appendix E lists all the modules with their locations (PL/I library or link library), lengths in bytes, and brief descriptions.

Private Libraries

JOB LIBRARY

If you want to execute programs that are not used frequently enough to justify their inclusion in the link library, you can identify the private library that contains them as a job library by naming it in a special form of DD statement with the name JOBLIB. You must place the JOBLIB statement between your JOB statement and the first EXEC statement of your program. The control program will then respond to a request for the execution of a program by first searching the job library (unless the program already exists in main storage); if the program is not in the job library, the control program will then refer to the link library in the normal way.

The JOBLIB statement must contain at least two parameters, DSNAME and DISP. You must code the DISP parameter as DISP=(NEW,PASS), DISP=(OLD,PASS), or DISP=(SHR,PASS) to ensure that the job library remains available throughout the job; however, if you code DISP=OLD or DISP=SHR, the job scheduler assumes DISP=(OLD,PASS) or DISP=(SHR,PASS). If you code DISP=(NEW,PASS), you must also specify the amount of space required by including the SPACE parameter. If the private library is not cataloged, you must identify

the volume that contains the library, and the unit that will process it (UNIT and VOLUME parameters).

You can concatenate two or more private libraries in a single job library by defining them in separate DD statements and leaving the name field of all but the first statement blank. The control program will search these libraries in the order of appearance of the DD statements that define them.

Figure 12-8 illustrates the execution of the load module created in Figure 12-5 from the private library FLM; in this example, the program lists the names of the members of the link library.

STEP LIBRARY

You can identify a private library as a step library in a similar manner to that described above for a job library. A STEPLIB DD statement applies to a job step only and overrides any JOBLIB DD statement for the duration of the job step. You can also specify a step library (but not a job library) in a cataloged procedure.

```
//J062PGE3 JOB
//JOBLIB DD UNIT=2311,VOLUME=SER=D186,DSNAME=FLM,DISP=OLD
// EXEC PGM=DIRLIST
//LINK DD DSNAME=SYS1.LINKLIB,DISP=SHR
//SYSPRINT DD SYSOUT=A
```

Figure 12-8. Use of JOBLIB Statement

Chapter 13: Multitasking

Introduction

Multitasking is the term used in PL/I to describe the concurrent existence of several tasks within the execution job step. Multitasking should not be confused with multiprogramming, which is the control of independent jobs running concurrently. PL/I multitasking can be used only under an MVT system whilst multiprogramming can be run under MFT or MVT controlled systems.

A multitasking program will usually contain one or more procedures within a main procedure, each of which can have a different priority, each identified by different task names and each capable of calling or being called by each other. For example:

```
A: PROC OPTIONS(MAIN,TASK);
   CALL B TASK(T1);
     .
     .
     .
B: PROC;
   CALL C TASK(T2);
     .
     .
     .
C: PROC;
     .
     .
     .
   END C;
   END B;
   END A;
```

A multitasking program, using the multitasking facilities provided by PL/I, may enable you to make fuller and more efficient use of machine time by reducing the time during which the CPU is waiting or the input/output devices are not used. It is essential, however, to understand the concepts involved and to arrange your program accordingly. This chapter sets out to aid your understanding of this feature; multitasking is also discussed in detail in IBM System/360 Operating System: PL/I (F) Language Reference Manual.

Multitasking Requirements

SYSTEM/360 REQUIREMENTS

For a full discussion of these requirements, see Appendix D, 'System Requirements.'

OPERATING SYSTEM REQUIREMENTS

PL/I multitasking uses the MVT system. A PL/I program compiled with the TASK option and executed under the MFT or PCP systems will terminate abnormally. All the tasks exist within the same job step; the priorities for these tasks exist only within the job step and can be varied over a range determined by the job step priority.

The minimum PL/I multitasking overhead in main storage and execution time, over and above that resulting from a single-task PL/I program, is:

Main storage:	about 3.5K bytes plus 2K bytes for each subtask attached.
Execution time:	about 70 milliseconds per task attached (model 40).

PROGRAMMING REQUIREMENTS

Compiler Level

Multitasking requires a PL/I (F) compiler and a PL/I subroutine library of version 4 level or later. Programs based on earlier verions can only be executed in a multitasking environment if recompiled with the TASK option (see below); even then subprograms in these programs may need rearranging to execute successfully.

Procedure Options

All programs and external compilations that are to be executed in a multitasking environment must have been compiled with

the TASK option in the external PROCEDURE statement. For example:

```
X: PROC OPTIONS(MAIN,TASK);
```

A CALL statement with the EVENT, PRIORITY, or TASK options also requires a multitasking environment; you should, therefore, specify the TASK option for the external procedure. If you do not, it is assumed by default.

If you use the TASK option, the PL/I library multitasking modules, instead of the single-task modules, will be link-edited into the load module. The load module thus contains multitasking modules only when multitasking is required.

Combination with Other Languages

When a routine in IBM System/360 assembler language is to be used in multitasking, the DSA obtained in it must be at least 108 bytes long. (The minimum length for a DSA in a non-multitasking environment is 100 bytes.) PL/I library routines used in multitasking must be those designed for multitasking, for example, IHEITH, IHETSA.

Multitasking Management

There are a number of topics that must be fully understood for successful multitasking management. These are:

1. Transfer of control
2. Use of priorities in PL/I
3. Programming considerations
4. Input/output handling
5. Task termination

TRANSFER OF CONTROL

One of the major differences between multitasking and non-multitasking programs is that of control over the CPU and input/output devices. In a non-multitasking environment the invoking procedure relinquishes control to the invoked procedure; in multitasking, an additional flow of control is established, so that procedures can be executed (in effect) concurrently. This means constant competition for control of the CPU, with control passing to procedures in order of priority or position in the system queue. A task is always the execution of a PL/I procedure. A task is always associated with one particular procedure; on the other hand, a procedure can be associated with several tasks. Because control can pass from task to task, with several tasks active at the same time, a procedure can be executed by several tasks concurrently. Figure 13-1 shows the transfer of control within one job, and how the CPU and the input/output devices are used.

Two points have been assumed to simplify the figure. One, that the various tasks within the program relinquish and request CPU and input/output devices at exactly the right moment; two, that no other job is trying to assume control at the same time. Obviously, in normal use, these two factors will play a part in the efficient execution of your program, and a discussion on the methods used to lessen their effect will be found under 'Use of Priorities' and 'Input/Output Handling' in this chapter.

A brief description of the changes in control, at the points indicated at the left of the figure, follows:

1. T2 obtains control of CPU due to its priority (3). T1 and T3 go into a WAIT state.
2. T2 relinquishes control of CPU and accesses input/output devices. T1 obtains control of CPU over T3 because it has a higher position in the system queue than T3 (still in WAIT state).
3. T1 relinquishes CPU and, wishing to access input/output devices, goes into a wait state (T2 still controls input/output). T3 gains control of CPU.
4. Whilst in control of the CPU, T3 attaches T4 and passes control to it,
5. T2 relinquishes input/output which is accessed immediately by T1 (waiting on input/output) and regains control of the CPU, having higher priority than T4. T4 goes into a wait state on CPU.
6. T1 relinquishes input/output control by way of a program instruction (e.g., WAIT (T3)) and T2 immediately accesses input/output, relinquishing control of CPU. T4 retains control of CPU.
7. T4 completes, returns to T3 which relinquishes control of CPU.
8. T3 releases input/output control to access CPU which has been relinquished by T2 on a program instructuon (e.g.,

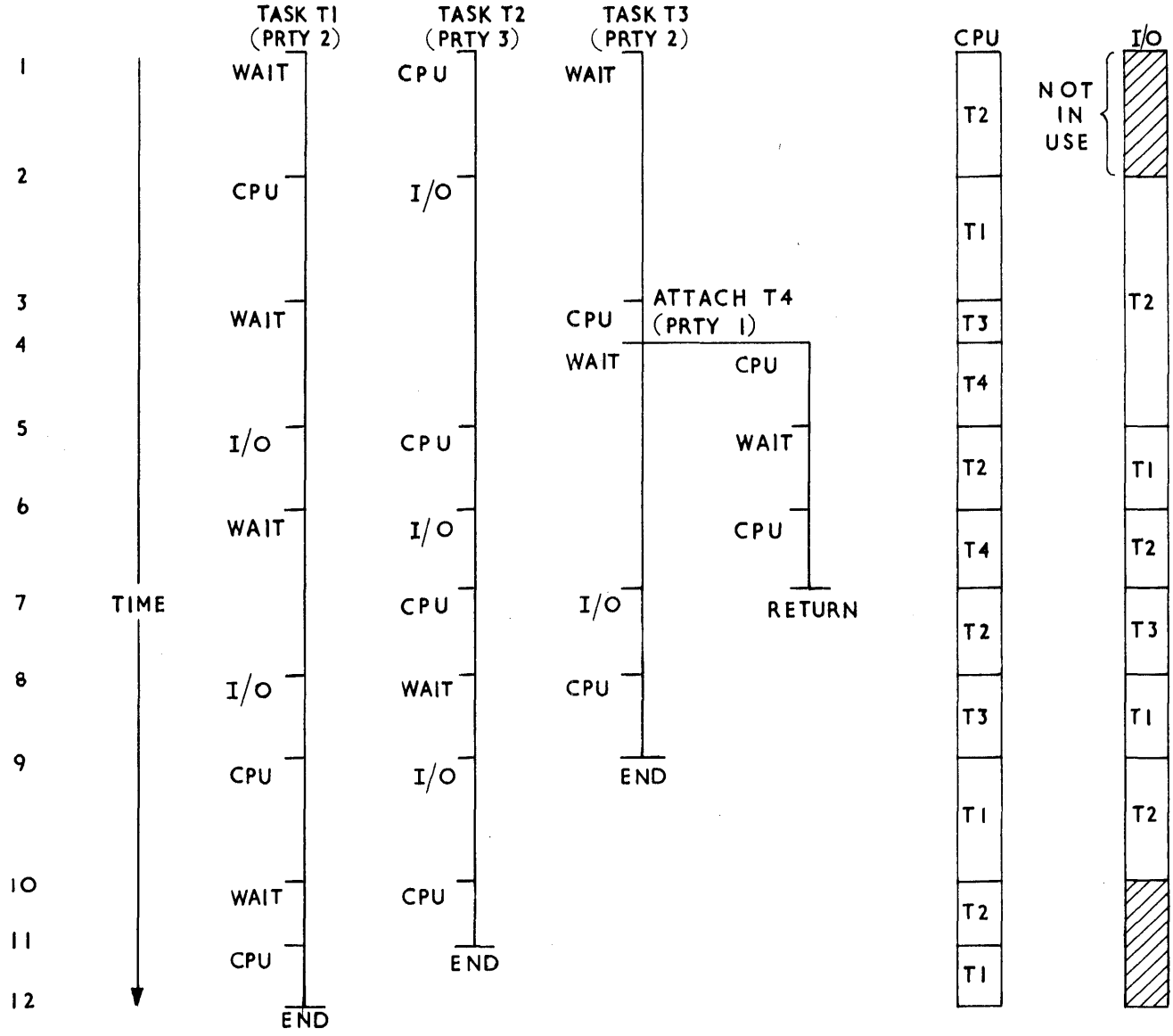


Figure 13-1. Transfer of Control within a Multitasking Program

WAIT (T3)). T1 accesses input/output again.

9. T1 obtains control of CPU when released by T3 (at end of task T3) and T2 once again accesses input/output.

10. T2 demands control of CPU by virtue of its higher priority and T1 goes into a wait state.

11. T2 relinquishes control of CPU and ends; T1 gains control of CPU.

12. T1 reaches end of task.

The columns to the right of the chart show the use of the CPU and the input/output devices. You can see that, with careful planning, the CPU is in use continuously. In this simple case, only one input/output channel has been indicated, thereby creating some WAIT situations which would not occur if more than one channel was used.

You should note the correlation between CPU control and the use of input/output devices. If a task gains control of the CPU and immediately goes into an input/output operation, control of the CPU

reverts to the task with the next highest priority, or to the task next in the system queue, including tasks in other jobs in different regions. It is therefore possible, by lack of planning, to gain control several times during the course of your program only to lose it immediately each time, and, finally, have to wait for another job to finish execution before you can continue.

Control of the CPU is also relinquished if control is requested by a task which has a higher priority than that of the task already using the CPU. The task thus displaced must once again compete with other tasks on the basis of priority or queue status.

The control of input/output devices, however, whilst conforming to the rules of priority and queue status for access, does not interrupt an operation to transfer control. If a task requests control of input/output devices whilst those devices are in use, the second request is initiated but control is not passed until the first operation has been completed.

USE OF PRIORITIES IN PL/I

Each task existing within a main procedure in the same execution job step can have a different priority. By evaluating the priority of a task, the system decides which task shall obtain control of the CPU. Tasks with equal priorities are evaluated on their relative positions in the system queue.

You can specify the priority associated with a particular job using the PRTY parameter of the JOB statement. The job step priority can have any value from 0 to 14 inclusive, the higher the value the higher the priority.

You can specify the job step priority of a job by using the PRTY parameter of the JOB statement, or the DPRTY parameter of the EXEC statement.

This priority determines the initial priority of the major task in the PL/I program, using the formula:

p=(16*(job priority)) + 10

The absolute range of values provided by this formula is 10 through 234; the range for a particular program is from 10 through a value that depends on the job priority. This value is not only the initial priority of the major task, it is also the maximum priority that any other task in the job can have. If an attempt is made to create a task priority greater than the maximum priority, the task will be executed at the maximum priority.

The priority of any task can be reduced to zero; an attempt to create a priority of less than zero will result in the task concerned being excuted at zero priority. An attached task can have a priority greater or smaller than its attaching task, provided that this priority is within the limits given above. A priority can be changed within a program by any other task.

The value of the priority for a task when the task is created is known as the despatching priority. The despatching priority of the attaching task can be equal to or less than the maximum priority but never greater than it, and can be changed during execution; this will affect the despatching priority of a subtask created after the change. The despatching priority of a subtask can also be changed during execution.

A priority can be assigned to a task variable before the variable is associated with an active task. If the task is attached without the PRIORITY option, the priority has the value that was assigned to the task variable.

These conventions must be interpreted carefully when the PRIORITY pseudo-variable, built-in function, or option is being used to manipulate the priority of a task. In particular, the effect of the maximum priority in restricting the manipulation should be noted.

In the example shown in Figure 13-2, the priority is established by:

```
//jobname JOB123,J.SMITH,MSGLEVEL=1,PRTY=2
```

Hence maximum priority=42. It is assumed that none of the omitted statements can cause transfer of control or create a wait situation. Control passes through the program to statement 12. Here task T1 is attached with a priority of 45; this is greater than the maximum priority, so T1 is given the maximum priority, 42. As this is higher than that of T(Z), the major task, control at once passes to T1, that is, to statement 26.

At statement 31, task T2 is attached with a priority of 38. Control remains in T1; statement 32 is executed. The priority of T2 is now the highest priority (T(Z)=35, T1=32, T2=38), so control passes at once to statement 41. After the execution of statement 47, the tasks have the priorities

Statement No.		Statement	Priority or Value
1	Z:	PROC OPTIONS(MAIN,TASK);	T(Z)=42
		.	
		.	
		.	
6		PRIORITY=-7;	
		.	
		.	
		.	
12		CALL Y TASK(T1) PRIORITY(10);	T(Z)=35
13		.	T1=42
		.	
		.	
16		PRIORITY=2;	T(Z)=37
17		A=PRIORITY(T1);	A=-5
18		PRIORITY(T1)=-9;	T1=28
		.	
		.	
		.	
25		END Z;	T(Z)=37
26	Y:	PROC;	T1=42
		.	
		.	
		.	
31		CALL X TASK(T2) PRIORITY(-4);	T1=42
			T2=38
32		PRIORITY=-10;	T1=32
33		B=PRIORITY(T2);	NE
		.	NE
		.	NE
		.	NE
40		END Y;	NE
41	X:	PROC;	T2=38
		.	
		.	
		.	
47		PRIORITY=-8;	T2=30
		.	NE
		.	NE
		.	NE
55		END X;	NE

Note: T(Z) is the priority of the major task, NE=not executed.

Figure 13-2. Flow of Control through a Program

T(Z)=35, T1=32, T2=30. Control now returns to T(Z), at statement 13.

In this example, the main procedure Z terminates normally, while the procedures Y and X terminate abnormally. This is because:

1. The value of the PRIORITY pseudo-variable specified in procedures Y and X is such that it causes control to be transferred out of these procedures, leaving several statements in the procedure unexecuted.
2. There is no statement that causes either procedure to be reentered before the main procedure is terminated.

Only the priorities of the current task and of an immediate subtask of the current task can be determined directly. The calculation of priorities in PL/I is relative, not absolute; absolute values, including that of the major task, are not easily available.

PROGRAMMING CONSIDERATIONS

A task may lose control under any of the following circumstances:

1. Termination.
2. Input/output operations.

3. A task is attached with a higher priority than the current task.

4. Use of the PRIORITY pseudo-variable.

5. A higher-priority task may come out of a wait state or may complete input/output.

6. Use of the DELAY or WAIT statements.

Event Option

Only one task at a time can wait on a particular event. If a situation occurs where two or more tasks wait on the same event, a diagnostic message is provided. This kind of situation can easily occur; for example:

```
CALL X TASK(A);

CALL X TASK(B);
```

If no event is explicitly declared in X, then, if X contains a WAIT statement, both tasks may be waiting for the same event.

Variables

The scope of variables must be watched carefully in a multitasking program. A variable that has not been explicitly declared may be altered unpredictably when control passes from one task to another. For example:

```
          .
          .
          .
        COMPLETION(ASSIGNMENT)='0'B;
        A=1;
        CALL A_TEST(A) TASK(ONE);
        WAIT(ASSIGNMENT);
        A=2;
        CALL A_TEST(A) TASK(TWO);
A_TEST: PROC(PARM);
        DCL VARIABLE FIXED BINARY;
        VARIABLE=PARM;
        COMPLETION(ASSIGNMENT)='1'B;
        IF VARIABLE=1 THEN DO;
            .
            .
            .
```

The WAIT statement ensures that the program must wait for the task to assign a value to a variable known only to A_TEST, and then set the even 'ASSIGNMENT' complete. If you did not use the WAIT statement, TASK(ONE) PARM would be overwritten by TASK(TWO) PARM; in which case, A possibly may have the value 2 before the assignment to VARIABLE. If this was so, the statements following THEN DO would never be executed.

ON-Units

If there is an ERROR ON-unit in a task, and the condition to which the ON-unit applies is raised in a subtask of that task, a GO TO out of the ON-unit will also raise the ERROR condition, causing the ERROR ON-unit to be reentered repeatedly in a loop, because the GO TO label will not be known to the attached subtask. You can avoid this problem by using a separate ERROR ON-unit in the subtask with a GO TO label that is known to the subtask. For example:

```
MAJOR:  PROC OPTIONS(MAIN,TASK);
        ON ERROR GO TO FINISH;
        CALL A TASK;
            .
            .
            .
A:      PROC;
        ON ERROR GO TO FINISH;
            .
            .
            .
        END A;
            .
            .
            .
FINISH: END MAJOR;
```

CHECK Condition

If, in a multitasking program, the use of the CHECK condition is not carefully synchronized, the results obtained may be unpredictable. A program executed with the CHECK condition may produce different results to the same program not using CHECK, since the presence of CHECK in a program may cause the task in which it occurs to wait for input/output and hence to lose control. When this task eventually regains control, some of the variables may have new values

SNAP Option in ON-Unit

If an entry point has been called with the TASK option specifying a task name, the entry point in the SNAP print-out is followed by the task name in brackets.

The handling of input/output in multitasking must be approached with some care. With the changes in control mentioned earlier and with several tasks active at the same time, the same data set may be accessed by different tasks; the same variable may be processed or the same ON-unit invoked. This could mean a record is overwritten, a variable may be changed inadvertently or, if a task terminates abnormally, either the record or the value of the variable may be lost. You may also lose control of the CPU at unexpected places in your program. To over come such difficulties, you should make use of the EVENT option, the WAIT statement, the DELAY statement, the EXCLUSIVE attribute, the COMPLETION and STATUS built-in functions (also usable as pseudo-variables for testing), in addition to the PRIORITY pseudo-variable already mentioned. There follows a brief description of each, with an example of when to use it and a note on any precautions to be taken.

EVENT Option

An input/output event must be waited for in the task that initiated it, for example:

```
     CALL A TASK(T1);
        .
        .
        .
A:   PROC;
     CALL B TASK(T2) EVENT(EV2);
```

Procedure A will not complete until procedure B has completed. You should remember, however, that an enforced wait of this nature wastes CPU time. You must, therefore, whenever possible, ensure CPU overlap with another task if the task waiting on an EVENT has to perform much input/output. (This applies also to the WAIT statement.)

Another instance of the difficulties met when using an input/output event occurs in the use of WRITE statements in REGIONAL(3) files. If a REGIONAL(3) file with U- or V-format records is opened for DIRECT UPDATE or OUTPUT, and records are being added to the file, then under certain conditions, the program may not execute properly. For example, if two or more tasks are simultaneously attempting to add records to the same track of the data set, and at least one of the WRITE statements concerned has the EVENT option, then if the WRITE statements are not correctly synchronized, the results are unpredictable.

You can avoid this difficulty in several ways:

1. Avoid using the EVENT option on such WRITE statements.
2. Confine such WRITE statements to one task.
3. Use non-input/output EVENT variables and WAIT statements to synchronize the WRITE statements in the various tasks.

WAIT Statement

This statement will cause a WAIT in the sequential processing of the task in which it appears. By using the WAIT statement you can ensure the completion of subtasks and the order of control of the CPU. For example:

```
P1:  PROC;
     CALL A EVENT(EV1);
     CALL B EVENT(EV2);
     WAIT(EV1,EV2,)(2);
       .
       .
       .
     END P1;
```

Using the WAIT statement as shown above also prevents the main procedure from completing whilst its subtasks are still active, a possiblity if A or B above had gone into an input/output operation immediately (assuming you omitted the WAIT statement). Control would have passed from P1 to A, from A to B, and from B to P1, where your program may have entered the routine to end P1. P1 would then have completed, leaving A and B still active.

The expression (2) in the WAIT statement is optional and in this case means P1 must wait on both A and B. If the expression had been (1), then P1 will only wait on A or B to complete before proceeding. A WAIT statement could have been placed between A and B in the above ensuring a step-by-step passage of control, with A not completing until B completed.

Note that the WAIT statement must specify event names not task names.

Use the WAIT statement with caution; it is relatively easy to cause errors in your program by its use. For example, having two tasks waiting on the same event, or by waiting on the wrong task. Refer to IBM System/360 Operating System: PL/I (F) Language Reference Manual for full details of the WAIT statement.

DELAY Statement

This statement allows a task to wait for a specified period without reference to an event variable. It causes suspension of execution by 'n' milliseconds, with 'n' expressed within the statement. The obvious precautionary note here is that you must know how long you wish to delay executuon.

EXCLUSIVE Attribute

This attribute, which you may specify only for DIRECT UPDATE files, prevents one task from interfering with an operation by another task, (see note below for further information) especially from one task overwriting a record on a data set used by another. (The EXCLUSIVE attribute only applies within one job. It is possible for two jobs (not tasks within one job) to access the same record at the same time through the use of two files, even though those files may be EXCLUSIVE.)

COMPLETION Built-in Function

This built-in function returns the current completion value of the event variable named in the argument. This value is '0'B if the event is incomplete, or '1'B if the event is complete.

STATUS Built-In Function

This built-in function returns the current completion value of the event variable named in the argument. This value is non-zero if the event variable has been set abnormal, or zero if it is normal. The non-zero value is set to 1 as a result of the completion of the task or input/output operation with which the EVENT variable has been associated by the EVENT option. If the non zero value is user-defined, it can be set to any value the user selects.

The following example, using a part of the example in the WAIT statement section, shows the use of these built-in functions.

```
P1: PROC;
      .
      .
      .
    CALL A TASK(T1) EVENT(EV1);
    CALL B TASK(T2) EVENT(EV2);
    WAIT(EV1,EV2)(2);
        .
        .
        .
A:  PROC;
      .
      .
      .
    END A;
B:  PROC;
      .
      .
      .
    END B;
    IF COMPLETION(EV1)=0
     THEN PUT LIST('TASK T1 NOT COMPLETE');
    IF STATUS(EV2)¬=0
     THEN PUT LIST('TASK T2 ABNORMAL')
      .
      .
      .
    END P1;
```

Note: You should acquaint yourself thoroughly with all aspects of all the input/output handling aids listed above before using them. Full details of their use can be found in IBM System/360 Operating System: PL/I (F) Language Reference Manual.

Synchronization

Input/output synchronization means avoiding operations on a given data set by two or more tasks at the same time. If synchronization is inadequate, then either a system completion code of 001 is returned or the results are unpredictable.

If a task is terminated abnormally while an input/output operation on a file is in progress in an attached task, and later the same file is accessed in another task, the results may be unpredictable.

System Completion Code 001: This code is returned if unrecoverable input/output error is caused by unsynchronized access to a data set from more than one task, or from more than one file.

System Completion Code 301: This code is returned under one of two circumstances:

1. Synchronization error in PL/I program. This occurs if an attaching task attempts an input/output operation on a file on which an attached task is

already performing an input/output operation. Use of EVENT and WAIT can eliminate this problem.

2. Attaching task is terminated abnormally while an attached task is still active. This occurs when an attached task is performing an input/output operation by means of QSAM and the attaching task is terminated abnormally (due to, for example, a source program error). The attaching task issues a WAIT to the ECB (event control block) associated with the file, in order to allow the input/output operation to complete before the file is closed. But the attached task has already issued a WAIT for the same ECB; as only one WAIT can be issued for an ECB, the operating system terminates the program abnormally and a completion code is returned.

Variables

You should ensure that two references to the same variable cannot be made at the same time. Again, this problem can be overcome by judicious use of WAIT statements. Subject to this qualification, and the normal rules of scope, the following rules apply:

1. You may refer to static variables in any task in which they are known.

2. You may refer to an automatic variable in any block in which it is known, to which it is passed as an argument, or in which it is referred to using a valid locator variable.

3. You may refer to a controlled variable in any task in which it is known. Allocation and the freeing of allocated storage are strictly bound by task boundaries. When a task is terminated, all allocations made within that task are freed.

4. Based variables again are closely bound by the task in which they are allocated. The allocations may not be known in the task referencing them.

Strings

You may obtain unpredictable results when two tasks are performing simultaneous operations on the same bit string or character string. Unpredictable results may also occur when an operation involving an unaligned bit string is taking place in one task at the same tine as an operation involving data, which is not necessarily bit data and whose storage is contiguous with that of the bit string, is taking place in another task. The occurrence of this problem is likely to be extremely infrequent. However, if it does occur, the WAIT statement and COMPLETION pseudo-variable should be used when multitasking to avoid such results. The following examples indicate cases which may produce unpredictable results because the attaching task and subtask are executing simultaneously.

Example 1:

```
MAIN: PROC OPTIONS(MAIN,TASK);
      DCL C CHAR(3) VAR INIT('1');
      CALL A EVENT(E);
      C=C||'2';
      WAIT(E);
A:    PROC;
      C=C||'3';
      END;
      PUT DATA(C)
      END MAIN;
```

In this example the execution of the PUT DATA(C) statement will give unpredictable results: the value of C may be '12' or '13' or '123' or'132'.

Example 2:

```
MAIN: PROC OPTIONS(MAIN,TASK);
      DCL 1 A,
          2 B CHAR(1) INIT('X'),
          2 C BIT(3) UNALIGNED INIT('111'),
          2 D FIXED DEC(2,0) INIT(0);
      CALL SUB EVENT(E);
      C='101'B;
      WAIT(E);
SUB:  PROC;
      B='Y';
      D=6;
      END;
      PUT DATA(A);
      END MAIN;
```

In this example the execution of the PUT DATA(A) statement will give unpredictable results because the values of B,C, and D may be incorrect.

SEQUENTIAL Files

Use of a SEQUENTIAL file other than SYSPRINT in more than one task can present difficulties. For example, a task may be interrupted in the middle of a PUT statement. If, in the task that gets control, another PUT statement is executed

on the same file, the result, at best, will be disordered fields in the output buffer and, at worst, will be changes to the internal control blocks so that the original task may not continue to execute properly and will probably terminate abnormally. Again, in the updating of a SEQUENTIAL RECORD file, a REWRITE will replace the last record read (or waited for) irrespective of which task read it. Suitable use of event variables and WAIT statements will avoid these situations, but the best solution is to keep all references to a SEQUENTIAL file to within a single task.

Only one file should sequentially create a data set on a direct-access device or a tape volume. If two or more files are writing on the same sequential data set on a direct-access device, each file will write on the data set independently of the others, and records written by one file will be overwritten by records from another file. Similarly, if two or more files attempt to write on a sequential data set on the same tape volume, the program will terminate abnormally. To avoid this, you should open a file in a common ancestor task of those using the file name.

In sequential-access operations, a particular data set should be referred to by only one file at a time in a program. Thus, in the MVT system, the records will be written separately, and will not be mixed or overwritten. If two or more files opened for sequential access refer to two data sets related by, for example, having the same magnetic-tape device or the same data set name, then the records will be overwritten.

Since error messages are written on the SYSPRINT file, it should be opened in the major task. Otherwise SYSPRINT will be opened in each task, and the error messages may be overwritten or lost.

The use of SYSPRINT for large STREAM files in multitasking is not recommended. The implementation uses system facilities to synchronize operations (PUT statements and error messages) on the file; the effect of this is to make PUT statements on the SYSPRINT file longer to execute than PUT statements on other PRINT files.

It is possible to fill the SYSPRINT file, with subsequent loss of data, due to continuous and severe use by one task. Similarly, if the SYSIN file is accessed by two tasks at the same time, data will be lost or disordered.

TASK TERMINATION

Normal Termination

If a task terminates normally with active subtasks, then:

1. An indefinite wait situation might be created. For example, a task K (with subtasks I and M) might itself be a subtask of a task A. If I and M contain events that are waited for in A then, if K is terminated normally while I and M are still active, the result is an indefinite wait in A.

2. A warning message (IHE577I) will be put out on SYSPRINT for each immediate active subtask of the task which is terminated normally.

You must be aware that this situation (normal task termination with active subtasks) can happen unexpectedly. For instance, a task that was not expected to lose control may do so by some implicit input/output, as in the following example:

```
A: PROC;
     .
     .
     .
   Y=0;
     .
     .
     .
   CALL B PRIORITY(10);
     .
     .
     .
B: PROC;
     .
     .
     .
   X=5/Y;
     .
     .
     .
   END B;
   END A;
```

When ZERODIVIDE occurs in B (by execution of the statement X=5/Y;), a considerable amount of input/output may be necessary to load dynamically the error-handling module to deal with the interrupt. This can allow A to gain control of the CPU; if A is terminated normally before B can regain control, B is terminated abnormally.

If a subtask is terminated when it has active subtasks, the completion value of these subtasks will be unchanged i.e., 0, and their status values will be set abnormally.

Abnormal Termination

In addition to the information on abnormal termination provided in 'Input/Output Handling' in this chapter, the following is relevant to multitasking:

1. If a STOP statement is executed in a subtask, the FINISH condition is raised in the subtask, not in the major task.

2. If the EXIT statement is executed in a subtask, the subtask is terminated, and the status of the event variable in the subtask is set abnormal (if it is not already so).

3. If the ERROR condition is raised in a major task, and there is no ERROR ON-unit or normal return from an ERROR ON-unit, then the FINISH condition is raised and the program is terminated.

4. If the ERROR condition is raised in a subtask, and there is no ERROR ON-unit or normal return fron an ERROR ON-unit, an error message is printed, but the FINISH condition is not raised. The subtask is terminated and the status of its event variable is set to abnormal.

If the operating system detects an error, such as 'No main storage available,' in a subtask, then the subtask is terminated, the status of its event variable is set abnormal, and a message describing the error is printed on SYSPRINT (if it is open, otherwise on the console). If the subtask has subtasks of its own, the status of these subtasks will be unchanged. The ERROR condition is not raised (because the task is already terminated before the PL/I library receives control), but, if a SYSABEND or SYSUDUMP card exists, a dump will be produced.

If a subtask is expected to terminate abnormally, either with a STOP statement or a CALL IHEDUMP statement, any task or event variables associated with the CALL statement that attaches the subtask must be either declared STATIC or EXTERNAL in the invoking task or must be declared in the main procedure. If this restriction is not complied with, the main procedure may terminate abnormally with a system completion code of 201 when the subtask is detached.

Multiprocessing

Multiprocessing permits two or more jobs to be executed simultaneously. On a machine with a single CPU, the highest priority task which is not in the wait state or not performing any input/output operations is executed by the CPU. No other tasks can be executed at the same time. On a multiprocessing machine (i.e., a machine with more than one CPU), the CPUs can execute tasks simultaneously.

The following example is assumed to be executed on a two-CPU machine with no other jobs executing at the same time, i.e., both CPUs are available for execution. In the example, the major task attaches task T1, which has a higher priority than the major task, at the first CALL statement. After the CALL statement has been executed, the major task executes simultaneously with T1 even though one major task has a lower priority. At the second CALL statement, task T2 is attached, again with a higher priority than the major task. In this case, after the CALL statement has been executed, T1 and T2, being the two highest priority tasks, execute simultaneously and the major task does not proceed unless T1 or T2 perform some input/output operations or goes into the wait state. Assuming this does not happen, when T1 completes, a CPU is available for continuing the execution of the major task. The major task then executes simultaneously with T2 until the WAIT statement when (assuming that T2 has not completed by this time) it waits for the completion of T2 and terminates itself. If the major task did not wait for T2 to complete, then T2 would be itself terminated abnormally when the major task terminated.

Example:

```
Z: PROC OPTIONS(MAIN,TASK);
     .
     .
     .
   PRIORITY=5;
     .
     .
     .
   CALL X TASK(T1) EVENT(E1) PRIORITY(2);
     .
     .
     .
   CALL Y TASK(T2) EVENT(E2) PRIORITY(2);
     .
     .
     .
```

```
X: PROC;
     .
     .
     .
   END X;
Y: PROC;
     .
     .
     .
   END Y;
     .
     .
     .
   WAIT(E2);
   END Z;
```

SYNCHRONIZATION

Synchronization of input/output is most important, since the likelihood of simultaneous operations on a given data set by two or more tasks is increased in multiprocessing. (See 'Input/Output Handling' in this chapter.)

Also, it is possible that an attaching task may complete execution before its subtask. Attaching a subtask with a higher priority than the attaching task does not ensure that the subtask conpletes first, since the attaching task may execute at the same time as the subtask although it has a lower priority. Hence, it is always advisable to wait for the termination of a subtask before terminating the attaching task.

Chapter 14: Other Facilities of the Operating System

Introduction

There are a number of IBM System/360 Operating System optional programs, supplied by IBM, that may be available at your installation. The optional facilities described in this chapter are those that can be called from a PL/I source program using interface routines provided by the PL/I library. They are:

Library Module	Facility Provided
IHEDUM	Dump of main storage
IHECKP, IHERES	Checkpoint/restart
IHESRT	Sort

You can invoke these library modules in your PL/I source program in exactly the same way as you can invoke one of your own subroutines, i.e., by a CALL statement specifying the relevant entry point and supplying any necessary arguments.

In addition to the optional facilities discussed in this chapter, many others are available that are not called from the PL/I program and are beyond the scope of this manual. The operating system utility programs are a set of programs suppplied by IBM, that perform a variety of housekeeping and support functions, such as listing the directory of a partitioned data set, copying and comparing data sets, and dumping and restoring the data contents of a direct-access volume. The utilities are invoked by EXEC statements, and are fully described in the publication IBM System/360 Operating System: Utilities.

The remainder of this chapter is devoted to a discussion of each of the facilities available by PL/I CALL, namely main storage dump, checkpoint/restart and sort.

Dump of Main Storage

The diagnostic ability of the PL/I (F) compiler is such that the majority of source program errors can be identified from the diagnostic messages generated at compile time or at execution time. Generally, therefore, there is little need for you to obtain a dump of main storage at execution time, as it will not add much to the information already provided by the diagnostic messages. The most useful information it will contain, as far as you are concerned, are the system and user completion codes, as they supply information about the job environment.

However, you should know how to obtain a dump, not only because the completion codes are useful diagnostic aids but because if your program terminates with a serious error that you cannot correct yourself, IBM programming support personnel may require a dump to help discover whether the error lies in the source program or in the compiler.

Three types of dump are obtainable; their characteristics are:

SYSABEND Dump: This contains the data in main storage and information about the data areas in use. The contents are:

1. Edited control information about system blocks. This information includes the completion codes.
2. Trace tables through supervisor calls (optional).
3. Contents of fixed-nucleus area.
4. Contents of dynamic area.

SYSUDUMP Dump: The contents are the same as a SYSABEND dump except that the trace tables and the contents of the fixed-nucleus area are omitted.

PL1DUMP: The contents of a PL1DUMP are:

1. Detailed information about the files and data areas used in each task.
2. Contents of the dynamic area.

The SYSABEND and the SYSUDUMP are produced when a job step terminates abnormally and the DD statements SYSABEND DD and SYSUDUMP DD respectively are present for the step. You can obtain a PL1DUMP dump by coding:

```
CALL IHEDUMx [(argument)];
```

in your source program, where 'x' is a suffix denoting the part of main storage to be written on a specified data set; the suffix also indicates whether or not processing is to continue after the dump. The four suffixes and their meanings are:

Suffix	Meaning
C	Dump the contents of that part of the main storage associated with the current task, then continue processing.
J	Dump the contents of that part of main storage associated with all active tasks, then continue processing.
P	Dump the contents of that part of main storage associated with all active tasks, then terminate the major task.
T	Dump the contents of that part of main storage associated with the current task then terminate this task.

The argument is optional and, if used, must be declared as ENTRY(FIXED BIN(31,0)). The argument is an expresssion that is evaluated at execution time; the result is a fixed binary integer that appears in the heading of the dump. This integer must be in the range 0 through 127. A number outside this range is replaced by 127.

For a PL1DUMP dump, a PL1DUMP DD statement must be supplied with the job step.

Dump Data Sets

If you want the dump to be written on a temporary or permanent data set for printing out later, code:

```
//SYSABEND DD parameters
```

where 'parameters' are the appropriate parameters for a temporary or permanent data set with CONSECUTIVE organization. This applies equally to PL1DUMP and SYSUDUMP DD statements.

Indicative Dumps

If you do not include a dump DD statement in your job input, then, if your program terminates abnormally, the result is:

Control Program	Dump
PCP or MFT	Indicative
MVT	None

An indicative dump provides a limited amount of control information, including the completion codes, but does not include the contents of either the fixed-nucleus or the dynamic areas.

The IBM-supplied cataloged procedures PL1LFCG, PL1LFCLG, PL1LFG, and PL1LFLG do not include dump DD statements for any job step. If you want to specify one, you must qualify the ddname with the step name, that is, code //GO.SYSABEND, etc.

Checkpoint/Restart Interface

This section briefly describes checkpoint/restart for the PL/I user, particularly the syntax and meaning of the associated PL/I CALL statements. If you intend to use this facility you should refer to IBM System/360 Operating System: Advanced Checkpoint/Restart Planning Guide, in which the facility is fully described.

The purpose of the checkpoint/restart facility is to allow you to save machine time in the event of a job-step failure. You can restart execution at the beginning of the job step or at a previously selected point within the step; in the latter case, the essential data for the restart is stored on a specified data set ready for use should a restart be necessary. The restart can be either automatic, in which case the job will continue after the failure (restarting at the designated point), or it can be deferred until the job is resubmitted; the resubmitted job will start at the designated point instead of from the beginning. The advantage of using this facility is that the machine time used for steps prior to the one that fails need not be written off as wasted; further, if the progress of the step can be checked at various logical points, execution can be restarted at the last checkpoint before the failure, thus minimizing machine-time wastage.

Job-step failure may occur for a number of reasons. There could be a program interrupt or the program output could be erroneous. A program interrupt might be the result of, for example, a machine interrupt, a program error, an operator error or an error caused by a concurrently executing program. Erroneous output could be the result of a program error or an error in the input data to the program. Whatever the cause, the effect is either:

1. The job step terminates abnormally, with or without system failure, or

2. The job step terminated normally, but the output is partly or wholly useless.

To recover from this, you can use checkpoint/restart to start the job step again or to start execution at a selected point within the job step. To perform the latter, you must have included, at one or more points in your PL/I program, code that will cause job information to be recorded on a specified data set. The job information consists of the contents of the area of main storage used by your program and of certain system control data. The point in your program at which this information is recorded is a checkpoint; the data set on which it is written is a checkpoint data set.

The checkpoint taken can be either a single checkpoint or it can be one of a number of multiple checkpoints. The difference is:

Single checkpoints: The information written at a checkpoint overwrites the information written at any previous checkpoint. Only the latest checkpoint information is currently available.

Multiple checkpoints: The information written at a checkpoint is placed on the data set after the information written at previous checkpoints. The information taken at all the checkpoints is currently available.

Thus, with a single checkpoint, restart within the job step can be made only from the last checkpoint taken. With multiple checkpoints, restart within the job step can be made from any checkpoint.

The JOB, EXEC, and DD statements of JCL and the CALL statement of the PL/I (but not necessarily all of these items) may be used to specify the checkpoint/restart facility, depending on your requirements. The use of these statements for checkpoint/restart is described later in this section.

TYPES OF RESTART

A restart can occur automatically as soon as the job step has terminated, or it can be deferred and then attempted in a later job. Restart can be at the beginning of a step or at a checkpoint within the step. As a result, four types of restart are available:

Automatic step restart
Automatic checkpoint restart
Deferred step restart
Deferred checkpoint restart

Automatic Restarts

The current job is not terminated; immediately the step is terminated it is restarted in one of the following ways:

Automatic step restart: At the beginning of the step

Automatic checkpoint restart: At the last checkpoint taken

Automatic restarts can be initiated by the checkpoint/restart facility only when the step has been terminated abnormally with a completion code specified at system generation as eligible for automatic restart (this includes system FF3 abnormal termination after system failure). Automatic step and checkpoint restarts can be requested for the same step; the step restart remains in force until the checkpoint restart is invoked. An automatic restart requires action by the operator before it can occur; therefore, if you want an automatic restart you must inform the operator when submitting the job. Under the MFT or MVT control programs, the operator can defer an automatic restart if such a restart is not immediately convenient.

Note: Automatic checkpoint restarts are always taken from the last checkpoint written, regardless of whether single or multiple checkpoints are taken.

Deferred Restarts

The current job is terminated: it may be resubmitted later and restarted at the failing step in one of two ways:

Deferred step restart: At the beginning of the step

Deferred checkpoint restart: At a specified checkpoint (single or multiple checkpoints can be taken)

Deferred restarts can occur irrespective of whether the step has been terminated normally or abnormally. The action taken by the operator is that normally required for initiation of step execution.

CHECKPOINT/RESTART REQUIREMENTS AND DIAGNOSTIC AIDS

Checkpoint/restart can be used with any of the control programs PCP, MFT, or MVT. It requires some or all of the following items to be coded in the job stream:

Job control language:

JOB statement:	RD parameter RESTART parameter
EXEC statement:	RD parameter (if not in JOB statement)
DD statement:	For checkpoint restarts, a checkpoint data set is required.

PL/I

```
CALL IHECKPx
```

where 'x' is a suffix denoting an entry point. The PL/I library module IHECKP invokes the standard operating system CHKPT macro, which takes the checkpoint.

```
CALL IHERESx
```

where 'x' is a suffix denoting an entry point. The subroutines in this PL/I library module can invoke restart, or suppress checkpoint restart.

The use of these features and the syntax required for each of them, are described in this section.

The checkpoint/restart facility also includes various diagnostic aids. These are:

1. CHKPT return codes. When the CHKPT macro has been executed, a return code to indicate the result of the execution is provided and is returned by the IHECKP module to a user-defined field declared as a variable in the PL/I source program.
2. Completion codes. When a job step is abnormally terminated, system or user completion codes may appear on the operator's console.
3. Diagnostic messages. When a checkpoint is taken or a restart is requested, diagnostic messages specifying required operator or programmer actions may appear on the operator's console. These messages usually include a code that indicates the kind of action required.

The format of these messages and codes, and the interpretation of the values that appear in the various codes, are fully described in IBM System/360 Operating System: Advanced Checkpoint/Restart Planning Guide.

JOB CONTROL LANGUAGE DETAILS

The RD Parameter

The RD (Restart Definition) parameter is used to request an automatic step restart and (optionally) to suppress execution of the CALL IHECKPx statement. When coded in EXEC statement, it applies to the corresponding job step. When coded in the JOB statement, it applies to all job steps in that job and overrides an RD parameter coded in any EXEC statement in that job. The parameter syntax is:

```
RD[.procstep]={R|NC|NR|RNC}
```

where:

R(restart)	Requests automatic step restart.
NC(no checkpoint)	Suppresses the execution of the CALL IHECKPx statement and the implicit automatic checkpoint restart.
NR(no restart)	Checkpoints may be written but automatic restart is prohibited.
RNC(restart but no checkpoint)	Requests automatic restart but suppresses the execution of the CALL IHECKPx statement.

Notes:

1. If RD=value is coded in an EXEC statement for a cataloged procedure, it applies to all the steps within the procedure, and overrides any existing RD parameter in the procedure. If RD.procstep=value is coded, it applies only to the specified procedure step; this format can be used for each step in the procedure, in procedure-step order.
2. The CALL IHECKPx statement causes a request for automatic checkpoint restart and overrides an RD=R parameter.

3. RD=NC is provided to prevent (for a particular job) checkpoints being taken when a program that includes a CALL IHECKPx statement is being executed. If there is no CALL IHECKPx statement, RD=NC has no effect.

4. RD=NR is provided to prevent automatic checkpoint restart, in anticipation of a deferred checkpoint restart. If there is no CALL IHECKPx statement, RD=NR has no effect.

5. RD=RNC is provided to prevent (for a particular job) checkpoints being taken when a program that includes a CALL IHECKPx statement is being executed. If there is no CALL IHECKPx statement, RD=RNC is treated as if it were RD=R.

6. If no RD parameter is specified, automatic step restart is prohibited, but automatic checkpoint restart is allowed if the program includes a CALL IHECKPx statement.

The RESTART Parameter

The RESTART parameter is coded in the JOB statement of a job that is to be submitted for deferred step or checkpoint restart. For step restart, it specifies the step at which execution will be restarted. For checkpoint restart, it specifies the checkpoint at which execution is to begin, and the step in which this checkpoint exists. The parameter syntax is:

```
RESTART=({stepname                    }[,checkid])
         {stepname.procstepname       }
          *
```

where:

stepname	Is the name of the step at which execution is to begin.
stepname. procstepname	Is the qualified name of the restart step when this step is a step of a cataloged procedure.
*	Indicates that restart is to begin at the first step. This could be the first step of a cataloged procedure.
checkid	Is a character string identifying the checkpoint at which execution is to begin. It consists of up to sixteen characters from the character set available. Special characters must be enclosed in single quotes; single quotes in the string must be represented as double quotes. The checkid is omitted for step restarts.

The DD Statement for a Checkpoint Data Set

This is always required for a checkpoint restart. It must be included in the job stream when the checkpoint data set is created, and also when a checkpoint restart is to be executed. The checkpoint data set must be either a sequential or a partitioned data set; see Chapters 3 and 12 respectively for information on the creation of these types of data sets.

Creation of Checkpoint Data Set: The DD statement for this data set is included with the other DD statements for the job step. The essential parameters are:

1. DSNAME: Any name can be used. A dsname need not be specified for automatic checkpoint restart, as the data set used here need only be a temporary one. A dsname is always required for a deferred checkpoint restart. If you want to create a partitioned data set, the dsname should be specified, but the member name should be omitted.

2. UNIT: A magnetic-tape device or any direct-access storage device can be specified.

3. VOLUME: Nonspecific volume requests can be used for automatic checkpoint restart; for deferred checkpoint restart, the preferred reference is VOLUME=SER=volnumber. If the checkpoint data is to be written on a multivolume data set, the serial number of the volume on which the data for a particular checkpoint is written is put out on the operator's console. You must obtain this number, as it must appear in the VOLUME parameter when execution is restarted at the required checkpoint.

4. SPACE: For direct-access devices, the space allocation must be that for a sequential or a partitioned data set. You may request secondary space (by the increment subparameter), but it will not be used. If 'end-of-volume'

(no more primary space) is encountered while writing a checkpoint on a direct-access volume, two actions are possible:

a. If you requested secondary allocation, the allocation is performed, and the checkpoint routine issues a return code of 8. This is returned to the PL/I program by IHECKP. The allocated space is not used.

b. If you did not request secondary allocation, the system executes an ABEND macro instruction applying to the step. The ABEND causes a system completion code D37 to be issued. This means that the step cannot be restarted. Thus, even though secondary space will not be used, you should specify secondary allocation to avoid abnormal termination. The amount of space required could be as much as one and a half times your partition or region size. It can be calculated from formulas given in IBM System/360 Operating System: Storage Estimates.

5. LABEL: If the checkpoint data set is on tape, the tape can have standard or nonstandard labels, or no labels.

6. DISP: The first subparameter should be NEW for single checkpoints, and must be MOD for multiple checkpoints (otherwise the checkpoint entries will always be overwritten). For automatic checkpoint restart, the second subparameter should be DELETE; for deferred checkpoint restart, it should be KEEP or CATLG.

7. DCB: If the checkpoint data set is to be written on a 7-track tape, you must code TRTCH=C.

The CALL IHECKPx statement causes the checkpoint data set to be opened and closed at the appropriate times. You must not include OPEN and CLOSE statements for this data set in your program. If you do, any attempt at restarting from a checkpoint will fail, and a system message will be given indicating that the checkpoint could not be found. The facility for opening a checkpoint data set in a problem program before taking checkpoints, as described in IBM System/360 Operating System: Advanced Checkpoint/Restart Planning Guide, is not available to PL/I (F) programs. Also, you need not declare the checkpoint data set as a file in your PL/I program.

For automatic checkpoints, you can specify a ddname for the checkpoint data set by means of the CALL IHECKPx statement. If you do not want to specify a ddname, the default ddname SYSCHK is applied.

Restarting at a Checkpoint: For automatic checkpoints, the checkpoint data set is both created and used for restart in the same job step. Therefore the parameters of the DD statement remain unchanged.

For deferred restarting at a checkpoint you must code the RESTART parameter RESTART=(stepname,checkid), on the JOB statement of the restart deck. This identifies both the step to be restarted and the checkpoint entry to be used to perform the restart. In addition, you must place a SYSCHK DD statement immediately before the first EXEC statement, and after any JOBLIB DD statements, in the restart deck. The SYSCHK DD statement must specify the checkpoint data set from which the checkpoint is to be read, and is additional to any DD statements in the deck that define data sets into which checkpoints have been, or will be written. The DD parameters specified must be the same as those used when the data set was created, except:

1. VOLUME: If the checkpoint data set is written on a multivolume data set, the volume on which the required checkpoint is written must be specified as the first volume to be requested.

2. DISP: The first subparameter must be OLD or SHR; the second either must be KEEP or may be omitted.

For more information on specifying the SYSCHK DD statement, and JCL requirements and restrictions for deferred checkpoint restart, refer to IBM System/360 Operating System: Advanced Checkpoint/Restart Planning Guide.

PL/I CALL STATEMENT DETAILS

The two library modules concerned with checkpoint/restart interface are IHECKP and IHERES. IHECKP establishes checkpoints for use when checkpoint restart is required. It is not used for step restart, since this facility does not require checkpoints and is specified entirely within the JCL for the job. IHERES can be used to cancel automatic checkpoint restart or to request a restart. Details of when and how these modules should be used are given below. Note that the effect of invocation of these modules could be nullified for a given job by means of JCL parameters (see 'Job Control Language Details' earlier in this section).

IHECKP (Checkpoint Module)

You can call this module from the PL/I source program in either of the following ways:

1. CALL IHECKPS (ddname, checkid, org, code);

2. CALL IHECKPT;

The difference between these two statements is that the first allows you to make a more comprehensive specification in your source program; the second is supported in the fifth version of the compiler mainly for compatibility with the previous version, which used an interim checkpoint/restart facility.

IHECKPS Entry Point

Execution of the statement:

```
CALL IHECKPS(...);
```

requests checkpoint/restart and causes the program information necessary for a restart to be recorded on a checkpoint data set. The module expects the arguments in the order ddname, checkid, org, and code. Thus, if you want to omit 'code', you can leave it out; but if you want to include 'checkid', 'ddname' must appear even if you would be satisfied with the default ddname. However, the first three arguments are all character strings, and defaults will be applied for any that evaluate to a null string. Details of the syntax, meaning, and default for the arguments are as follows:

ddname: This argument is any character-string expression (including variables and constants) that, when evaluated, yields a valid ddname, i.e., an alphanumeric character string not longer than eight characters, whose first character is alphabetic. (If the string is longer than eight characters, it will be truncated on the right.) This ddname is the name of a checkpoint data set; if the argument appears (and is not a null string), a DD statement with this name must appear in the job stream or the checkpoint will not be taken. If the argument is omitted, or is a null string, the default (SYSCHK) will be applied, in which case a SYSCHK DD statement must appear in the job stream.

checkid: This argument is any expression that yields a character string consisting of printable characters. Restrictions depend on the type of data set (sequential or partitioned) used to hold the checkpoint information. If the data set is partitioned, the ckeckid must be alphanumeric and its first character must be alphabetic; the length of the string should not exceed eight characters. For a sequential data set, the length should not exceed sixteen characters. In either case, if the string is too long it will be truncated on the right.

The argument identifies the checkpoint so that, when multiple checkpoints are used, a deferred restart can be requested to commence at a particular checkpoint instead of the last one taken (see 'The RESTART Parameter' earlier in this section). If this argument is omitted, or is a null string, a system-generated checkid will be used. To restart at a particular checkpoint, you will then have to refer to the console listing to find its identification.

org: This argument is any expression that yields a character-string value of length two, containing either 'PS' or 'PO'. It is used to define the organization of the checkpoint data set, 'PS' meaning sequential, and 'PO' meaning partitioned. If the argument is omitted, a default of sequential will be applied.

code: This argument is fixed-point binary integer variable of precision greater than 15. If the argument is specified, the checkpoint module will return a value indicating the result of the checkpoint request. The PL/I programmer can obtain the value by reference to the variable. The codes are those issued by the CKPT macro and are shown in Figure 14-1.

The facility provides a means of recognizing whether the program is being executed initially or as the result of a restart. You can test the variable at any point in the program and you may be able to by-pass the error that caused the step failure, by transferring control according to the value of the variable.

Note: It is advisable to declare IHECKPS as follows:

```
DECLARE IHECKPS ENTRY (CHAR(8)VAR,
      CHAR(16)VAR, CHAR(2), FIXED BIN(31));
```

to ensure that correct arguments are passed.

Code	Meaning
0	Successful completion
4	Restart occurred
8	Unsuccessful completion (program error)
12	Unsuccessful completion (input/output error)
16	Successful completion, but ENQs are outstanding and will not be restored on restart

Figure 14-1. Return Codes from Checkpoint Module IHECKP

IHECKPT Entry Point

Execution of the statement:

```
CALL IHECKPT;
```

requests checkpoint/restart and causes the program information necessary for restart to be recorded on the checkpoint data set. There must be a SYSCHK DD statement for this data set in the job stream. This entry point is included in the fifth version of the compiler for compatibility with the previous version; it is not as comprehensive as IHECKPS, and has the following restrictions:

1. The checkpoint data set must always be specified by the ddname SYSCHK.
2. You cannot identify the checkpoint in your program. To restart at a particular checkpoint, you would have to refer to the console listing to find its identification.
3. You cannot specify the checkpoint data set organization in your PL/I program.
4. Your program is unable to determine whether the current execution is the initial one or is the result of a restart (unless you pass the information to the main procedure, using the PARM field of the EXEC statement on deferred restart).

IHERES (Restart Module)

This module has two entry points, IHEREST (request for a restart) and IHERESN (cancelation of automatic checkpoint restart). Details of these entry points follow:

IHEREST Entry Point

Execution of the statement:

```
CALL IHEREST;
```

requests an immediate automatic restart of the job step. Its purpose is to accommodate those circumstances in which automatic restart does not occur even though an error has occurred that requires a restart. This happens in the case of program interrupts such as protection violation, when ABEND may not occur. (In any case, program check ABENDS are not eligible for automatic restarts.) IHEREST, in fact, forces an automatic restart by issuing an ABEND macro. This causes the program to terminate with a user completion code of 4092, which in turn causes a restart. This entry point can be used in, for example, an ERROR ON-unit for program interrupt errors (detected by an ONCODE value within the range 8091-8199).

Notes:

1. In order for IHEREST to cause a restart, your system programmer must have specified a user completion code of 4092 to be eligible for automatic restart when your operating system was generated. If this was not done, and your program calls IHEREST, it will be terminated with a 4092 completion code and no restart will take place.
2. Even if 4092 is an eligible completion code for automatic restart, calling IHEREST can still cause your program to be terminated with a 4092 ABEND and no restart, in the following cases:
   a. The RD parameter was not specified on your JOB or EXEC statement, and no checkpoints were taken.
   b. RD=NR or RD=NC was specified on your JOB or EXEC statement.
   c. RD=R or RD= RNC was specified on your JOB or EXEC statement, or checkpoints were taken and the operator replied 'NO' when the operating system issued a message asking whether your job should be restarted.

IHERESN Entry Point

Execution of the statement:

```
CALL IHERESN;
```

cancels a request for automatic checkpoint restart. If the RD parameter of the job statement is used to request an automatic step restart, that request will again be in

effect. The CALL IHERESN statement is equivalent to the CANCEL option of the CKPT macro, and is generally used to prevent repeated attempts to restart from a checkpoint (the CALL statement being executed or bypassed, depending on the return code held in 'code' argument to IHECKPS). Automatic restart can be reestablished by using a call to the checkpoint module. If a call to IHERESN occurs when automatic restart is already canceled, the call is ignored.

RESTRICTION ON USE OF CHECKPOINT/RESTART

The checkpoint facility operates in an operating system MVT environment but must be issued in the originating task with all other sub-tasks inactive. Specifying the TASK option using the PL/I (F) compiler, means that a checkpoint can only be issued in the control task which is not accessible to you. There is therefore the restriction that a checkpoint cannot be taken at all in a tasking program. A checkpoint may be issued in either the main or sub-procedures without the TASK option whether running under PCP, MFT or MVT.

There is also a further restriction that a checkpoint may not be issued between a DISPLAY(X) REPLY(Y) EVENT(Z) statement and the accompanying WAIT(Z) statement.

EFFECT OF CHECKPOINT/RESTART ON DATA SETS

The checkpoint routine preserves all the relevant information about all the data sets used by the step calling the checkpoint module. The treatment of various types of data set during restart is discussed fully IBM System/360 Operating System: Advanced Checkpoint/Restart Planning Guide; the following notes cover the most important points.

User Data Sets

1. Data sets that were open when the checkpoint was taken are repositioned, with the exception of data sets on unit-record devices such as card readers and printers.
2. Data set contents are not saved and restored.
3. If a file is opened with the UPDATE attribute and records are replaced after the checkpoint is taken but before restart, the contents at restart will not be the same as they were when the checkpoint was taken.
4. If multiple checkpoints are used, and restart occurs at a checkpoint other than the last one taken, data sets with MOD disposition or partitioned organization may be repositioned to a point other than that expected.
5. If a data set occupies multiple direct-access volumes and the checkpoint is taken before the end of one volume but restart occurs after the switch to the next volume, then the data set on the second volume must be deleted before restart; otherwise, when the time comes to switch to the second volume after restart, the system will not be able to do so, since a data set with a duplicate name will still exist on the second volume.

SYSIN/SYSOUT Data Sets

The data sets are handled differently from other data sets, depending on whether or not a multiprogramming system is being used. For further information, see IBM System/360 Operating System: Advanced Checkpoint/Restart Planning Guide.

Resident Access Methods and Checkpoint/Restart

The checkpoint/restart facility processes the checkpoint data set using BSAM or BPAM. The access method modules required to process the checkpoint data set must be resident in main storage. If they are not, and you attempt either an automatic or a deferred restart from a checkpoint, the restart program may fail with a program check attempting to read the checkpoint data set, and your job will be terminated. Full details on the modules required to be resident, and the procedure to make them resident, are described in, IBM System/360 Operating System: Advanced Checkpoint/Restart Planning Guide, and System Programmer's Guide, respectively.

Sort Interface

The PL/I(F) Compiler provides an interface between a PL/I program and the IBM System/360 Operating System Sort/Merge program. You invoke it by calling the PL/I library module IHESRT at the appropriate entry point; this in turn calls the Sort/Merge program by means of a LINK macro. When the sort is complete, control is returned from the sort program to the PL/I program through the IHESRT module; execution of the PL/I program is resumed at the point following the CALL IHESRT statement. If the sort program uses data sets directly for its input or output, all opening and closing of files is handled independently of PL/I.

Information defining how the data is to be sorted and the format of the records in which this data exists is passed as arguments to the IHESRT module.

The entry point to the IHESRT module that should be selected depends upon the source of the records to be sorted and their disposition afterwards. Four entry points are available:

IHESRTA Records in a data set are retrieved, sorted, and placed in another data set.

IHESRTB Records constructed or updated in a PL/I procedure are sorted and placed in a data set.

IHESRTC Records in a data set are retrieved, sorted, and passed to a PL/I procedure.

IHESRTD Records constructed or updated in a PL/I procedure are sorted and passed to a PL/I procedure.

Retrieval of records from a data set, passing them to the sort program, and placing the sorted records in a data set are all performed by the PL/I program.

Entry points IHESRTB, IHESRTC, and IHESRTD involve the use of the sort program user exits E15 and E35 to invoke a PL/I procedure that either supplies records for sorting or receives sorted records. It should be remembered that a PL/I procedure invoked from a user exit is invoked for each record passed to or received from the sort program. Each invocation involves a substantial time overhead necessary to establish a PL/I environment, such as restoring the PL/I error-handling facilities and allocating storage for automatic variables. However, a decision to use the PL/I sorting facilities should include consideration of such overheads in comparison with those that accrue from the use of SORTIN and SORTOUT data sets, their creation and retrieval time overhead, and the additional external storage requirements they create.

The records passed to the sort program are sorted until the required sequence is obtained. The sort is performed on the contents of selected fields within the record; up to sixty-four of these fields can be designated. This is adequate to ensure that even long, complicated records containing a small range of data types and values can be correctly and successfully sorted.

The sequence in which the records are placed is the IBM System/360 collating sequence.

Full details of the Sort/Merge program are given in IBM System/360 Operating System: Sort/Merge. Brief information on some aspects, for example, record format, storage requirements, and data set description, is provided here as a guide to the environment required, but this is not intended to supplant use of the Sort/Merge manual.

PL/I SORT ENVIRONMENT

Record Format

Blocked and unblocked fixed- and variable-length records can be passed to the sort program. Record size can vary over a wide range:

Minimum: 18 bytes

Maximum: about 32,000 bytes. The size for a particular application depends on the amount of main storage available and on the type of auxiliary storage used.

If the sort program reads or writes its records directly, its performance is improved if the input or output records are blocked. However, a PL/I procedure can process only one record at a time through a sort program user exit.

Storage Requirements

The minimum storage requirements for the sort program are:

Main storage: 16,000 bytes (PCP,MFT)
26,000 bytes (MVT)

Auxiliary storage: Three magnetic-tape units, or one direct-access storage device.

These minimum requirements are for the sort program when used without direct-access devices. Additional storage is required if sorting involves physical record lengths greater than 400 bytes, a large number of intermediate data sets, and the use of direct-access devices. The publication IBM System/360 Operating System: Storage stimates, Form GC28-6551, is intended to assist in estimating main storage requirements.

The sort program will work with the above minimum storage requirements, but may not provide the most efficient performance for a particular application. In general, the efficiency of the sort program improves as the main storage space it can use increases. The sort program will, if sufficient storage is available, select the most efficient sorting technique for a given application. The minimum storage requirements for the selection of the most efficient technique are:

Main storage: 24,000 bytes (PCP,MFT)
26,000 bytes (MVT)

Auxiliary storage: See Figure 14-2.

The devices used must not be mixed; either all tape units or all direct-access devices of the same type must be used. Tape units can be 7- or 9-track, or a mixture of both.

When direct-access devices are used, then sort performance is improved if:

1. Each data set is kept on a separate device.
2. The number of data sets used is a minimum.
3. All data sets are the same length.

Data Sets

Some or all of the following DD statements, in addition to the DD statements for the PL/I program, are required for a job step that uses both PL/I and the sort program. These DD statements are given in Figure 14.3.

The parameters for all but two of these data sets depend on the particular application requiring the sort. The two exceptions are:

```
//SORTLIB  DD DSNAME=SYS1.SORTLIB,DISP=OLD
//SYSOUT   DD SYSOUT=A
```

In an MVT environment, DISP=SHR should be specified in the SORTLIB DD statement.

PL/I sort checkpoints, if required, can be written on a data set identified by the DD statement //SORTCKPT DD ... etc. A deferred restart of a PL/I sort should use the DD statement //SYSCHR DD ... etc., to identify the checkpoint data set to the restart program. Further information on the use of checkpoint/restart is given in the section 'Checkpoint/Restart' in this chapter.

	Auxiliary-Storage Device			
	Number of Magnetic Tapes	Number of Data Sets		
		2301	2311	2314
Minimum storage for most efficient performance	4	3	3	3
Maximum storage permitted	32	6	6	17

Figure 14-2. Auxiliary Storage required for Sort.

ddname	Use of DD statement
SORTIN	Input to the sort program, not required if IHESRTB or IHESRTD is used.
SORTOUT	Output from the sort program, not required if IHESRTC or IHESRTD is used.
SORTWK01 SORTWK02 SORTWK03 . . . SORTWKnn	Data sets for sort program work space: at least three (six for a 2314) are required. 3i is the maximum that the sort program can use.
SORTLIB	Program library used by the sort program, always required.
SYSOUT	Data set used by the sort program for its diagnostic messages, always required.
SORTCKPT	Optional data set for checkpoints taken during sorting.

- Figure 14.3. DD Statements for Sort/Merge

If both the following occur:

```
//SYSOUT   DD  SYSOUT=A
//SYSPRINT  DD  SYSOUT=A
```

and the SYSOUT device is a magnetic-tape unit, you must take care that the two data sets do not use the same output device. This can be avoided by specifying a different device class for each data set.

DD statements for the sort program files SORTIN and SORTOUT are required only when the sort program obtains records for sorting directly from a data set, or when it writes the sorted records directly onto a data set. If they are required, there must not be any PL/I files with the same names or titles. The sort files are opened or closed by the sort program. SORTCKPT, SYSUDUMP, and PL1DUMP are optional, as is the number of SORTWKxx statements.

If you use a cataloged procedure for the job or job step in which sort is invoked, the appropriate DD statements for the sort program must be added to the appropriate procedure step.

If the sort program is in a private library, and not SYS1.LINKLIB, either a //JOBLIB DD statement or a //STEPLIB DD statement will be required for that private library.

USER CONTROL OF SORT DDNAMES

For multiple invocations of the sort program within a single job step, the standard ddnames SORTIN, SORTOUT, SORTWK and SORTCKPT can be changed by replacing the first four characters of the ddnames. This is achieved by adding an extra argument to the end of the argument list in the CALL statement. The argument is a character string of any length.

If the string is null, the standard ddnames remain unchanged. If the string is more than four characters long, only the first four characters are used. If the string is from one to four characters long, the first one to four characters in the standard ddnames are replaced.

The first character in the string must be alphabetic; otherwise, either a file-opening error may occur in the sort program or an error will occur during scheduling of the job step if an invalid ddname is found.

The PL/I (F) compiler does not permit a variable number of arguments to the same entry name in separate CALL statements. Therefore, a compilation which invokes multiple sorting operations must contain the CALL statements with the argument to specify the ddnames set to null ('') when using standard ddnames, and set to the required character string for the modified ddname.

Example:

```
TEST: PROC OPTIONS(MAIN);
          .
          .
          .
          .
          /*USING STANDARD DDNAME*/
      DCL STRING CHAR(2) INIT('PA');
      CALL IHESRTA(ARG1,ARG2,ARG3,
          ARG4,'');
          .
          .
          /*IN USING MODIFIED DDNAMES*/
      CALL IHESRTA(ARG1,ARG2,ARG3,ARG4,
          STRING);
          .
          .
          .
      END TEST;
```

In this example, the first invocation of IHESRTA requires the standard DD statements:

```
//SORTIN   DD  ...
//SORTOUT  DD  ...
//SORTWK01  DD  ...
//SORTWK02  DD  ...
```

In the second invocation of IHESRTA, the following modified DD statements are required:

```
//PARTIN   DD  ...
//PARTOUT  DD  ...
//PARTWK01  DD  ...
//PARTWK02  DD  ...
```

DEFINING THE SORTING APPLICATION

The information defining the data to be sorted and the records in which it exists is described in two sort program statements, the SORT statement and the RECORD statement.

The SORT Statement

The SORT statement describes the control fields within a record on which the sort is to be made. A SORT statement must be passed to the sort program as an argument of the CALL statement. Up to sixty-four control fields are permitted. The format of the SORT statement is:

```
SORT FIELDS=(b,l,f,s[,b,l,f,s]...)
            [,SIZE=m][,SKIPREC=2][,CKPT]
```

or alternatively

```
SORT FIELDS=(b,l,s[,b,l,s]...)FORMAT=X
            [,SIZE=m][,SKIPREC=x][,CKPT]
```

where:

b = first byte of the field to be sorted. Binary data can start on any bit within a byte and is specified, for example:

```
  7.2   Bit 2 in byte 7
 10.3   Bit 3 in byte 10
```

All other data starts on a byte boundary, which is specified as an integer.

l = length (in bytes) of the control field. Since binary data can start and end on any bit, its length is specified in the byte-bit notation given above. For example:

```
  2.1   the length of the binary data
        field is 17 bits
```

All other lengths are specified as integers.

f = data type. The code for the various data types is:

BI	Binary
CH	Character
FI	Fixed-point
FL	Floating-point
PD	Packed decimal
ZD	Zoned decimal

If all the control fields have the same data type, f can be omitted from the specification of individual control fields and FORMAT=x (where x is the data type) inserted after the right parenthesis and before the SIZE option, if used.

These sort data-type codes correspond to the following PL/I data types:

Sort	PL/I
BI	BINARY BIT
CH	CHARACTER
FI	FIXED BINARY
FL	FLOAT BINARY FLOAT DECIMAL
PD	FIXED DECIMAL
ZD	PICTURE

s = the order in which the contents of the field are to be sorted. The codes are:

A	ascending order
D	descending order

SIZE = m is the number of records to be sorted. If this number is not known precisely, an estimated total can be specified thus:

```
SIZE=Em
```

If a precise number of records is given, and more than this number is contained in the input to the sort, the sort will be terminated. If an estimated number is given or if the SIZE option is not specified and the intermediate storage is sufficient to contain all the records to be sorted, the sort will be completed; otherwise, the sort will be terminated.

SKIPREC = z is the number of records to be skipped before the sort begins. This allows a sort to begin at any point in a data set, omitting any records for which a sort is not required.

CKPT specifies that a checkpoint should be taken at several points in the sort program.

The specification of these sort control fields is subject to the following restrictions and conventions:

1. The total lengths specified for all the control fields in a SORT statement must not be greater than 256 bytes. If binary data specifies part of a byte, the whole of that byte must be included in the length count. For example, a binary field starting at 11.3 and ending at 27.2 is 17 bytes long.

2. All the control fields specified by the SORT statement must be in the first 4092 bytes of the record.

3. The maximum length of a decimal control field is 16 bytes; all other fields can be up to 256 bytes long.

The RECORD Statement

The RECORD statement describes the format and length of the records to be sorted. The format is:

RECORD TYPE=r, LENGTH=(l_1,l_2,l_3,l_4,l_5)

where:

r = record format. The code is:

F fixed-length
V variable-length

l_1 = length of each record in the input data set, as follows:

F-format: record length
V-format: maximum record length

The length must be the same as the LRECL value in the DCB parameter for the SORTIN data set; if it is not, the LRECL value is taken.

l_2 = length of each record to be handled by the sort program, as follows:

F-format: record length
V-format: maximum record length

If this value is not given in the RECORD statement, it is assumed to be equal to l_1.

l_3 = length of each record in the output data set, as follows:

F-format: record length
V-format: maximum record length

If this value is not given in the RECORD statement, it is assumed to be equal to l_2. The value must be the same as the LRECL value in the DCB parameter in the SORTOUT data set; if it is not, the LRECL value is taken.

l_4 = minimum length of V-format records in the input data set. If this value is not given, it is assumed to be the greater of:

1. The minimum necessary to contain the control fields specified in the SORT statement, or

2. The minimum physical-record length required by the operating system.

l_5 = the most frequently occurring record length in a data set containing V-format records. It is called the modal length. If this value is not given, it is assumed to be the average of the minimum and maximum lengths of the records in the input data set.

The specifications of these RECORD fields is subject to the following restrictions and conventions:

1. The lengths specified for V-format records must include the 4-byte count field at the beginning of each record.

2. When a direct-access device is used for auxiliary storage, the record length must not exceed one track.

3. The record format must be the same as that specified in the RECFM subparameter in the DCB parameter for the SORTIN and SORTOUT data sets. If it is not, the SORTIN RECFM specification is used.

4. Values in the LENGTH parameter that are equal to those assumed by the program can be dropped from the operand. Values dropped from the right-hand end of the operands are simply omitted; values dropped at the beginning or middle of the operand must be indicated by commas:

LENGTH=(l_1,l_2)
LENGTH=(l_1,,,l_4)

ENTRY POINT IHESRTA

Entry point IHESRTA is used for sorting records from one data set to another. The format of the CALL statement is:

```
CALL IHESRTA(argument1,argument2,
             argument3,argument4);
```

where:

- $argument_1$ = a character-string expression representing the SORT statement.
- $argument_2$ = a character-string expression representing the RECORD statement.
- $argument_3$ = an arithmetic expression that on evaluation gives a fixed-point binary integer of precision (31,0) specifying the amount of main storage available to the sort program.
- $argument_4$ = a fixed-point binary integer variable of precision greater than 15, that will contain the value of the return code returned by the sort program: 0 sort successful, 16 sort unsuccessful.

IHESRTA must be declared as an entry name with the appropriate parameters.

The value of the character-string expression for the SORT and RECORD statement has the form:

```
'bstatementb'
```

The blanks at the beginning and the end of the expression are always required. An embedded blank must occur between SORT and FIELDS, and between RECORD and TYPE; no other embedded blanks are permitted.

When character-string constants are used that are too long for one record of the PL/I source program, they are continued in the following record. You must take care that embedded blanks are not inadvertently inserted at the beginning and end of such records; the value of the SORMGIN parameter must be taken into consideration.

A PL/I program that uses IHESRTA requires both the SORTIN and SORTOUT DD statement, as well as any others that are necessary.

An example of such a program is given in Figure 14-4. This program sorts 80-byte records into an ascending sequence according to the alphanumeric data contained in two control fields. The major control field is in bytes 75 to 80; the minor control field is in bytes 16 to 21. The input records are obtained from the input stream. The sorted records are written onto a new temporary data set from which they are retrieved and printed by the PL/I program in job steps STEP3 and STEP4. Included in the output for this job are sort program diagnostic messages. These messages include the number of records that are in the input data set, and the number of records transmitted to the output data set. These numbers should be identical.

```
//R20A JOB
//STEP1 EXEC PL1LFCL
//PL1L.SYSIN DD *
   /* PL/I PROGRAMMING EXAMPLE USING IHESRTA */

   SORTA:  PROC OPTIONS(MAIN);

          /* ESTABLISH ENTRY POINTS TO THE SORT PROGRAM */

          DCL IHESRTA ENTRY(CHAR(35),CHAR(27),FIXED BIN(31,0),
                            FIXED BIN(31,0)),
              RETURN_CODE FIXED BIN(31,0);

         /* INVOKE THE SORT PROGRAM */

         CALL IHESRTA (' SORT FIELDS=(75,6,CH,A,16,6,CH,A) ',
                       ' RECORD TYPE=F,LENGTH=(80) ',
                       25000, /*MAIN STORAGE FOR SORT PROGRAM */
                       RETURN_CODE);

         /* TEST RETURN CODE */

         IF RETURN_CODE = 16 THEN PUT SKIP EDIT ('SORT FAILED')(A);
            ELSE IF RETURN_CODE = 0 THEN PUT SKIP EDIT ('SORT COMPLETE')
                                                       (A);
                 ELSE PUT SKIP EDIT ('INVALID SORT RETURN CODE.  CODE=',
                                     RETURN_CODE)(A);
          END SORTA;
/*
//STEP2 EXEC PGM=*.STEP1.LKED.SYSLMOD
//SYSOUT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SORTLIB DD DISP=SHR,DSN=SYS1.SORTLIB
//SORTWK01 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK02 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK03 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK04 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK05 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK06 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTOUT DD DSNAME=&&TEM,DISP=(NEW,PASS),
// SPACE=(TRK,(1,1)),UNIT=2314,DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SORTIN DD *
               AAAAAA                                                  AAAAAA
               ZZZZZZ                                                  ZZZZZZ
               444444                                                  444444
               CCCCCC                                                  ZZZZZZ
               XXXXXX                                                  999999
               CCCCCC                                                  888888
               CCCCCC                                                  AAAAAA
               CCCCCC                                                  ZZZZZZ
               CCCCCC                                                  ZZZZZZ
               333333                                                  ZZZZZZ
               VVVVVV                                                  AAAAAA
               EEEEEE                                                  AAAAAA
               333333                                                  ZZZZZZ
/*
//STEP3 EXEC PL1LFCL
//PL1L.DUMP DD DSNAME=&&GOSET(GO),DISP=(OLD,DELETE)
//PL1L.SYSIN DD *
  /* PL/I ROUTINE TO PRINT OUTPUT FROM SORT PROGRAM EXAMPLES */
  P:  PROC OPTIONS(MAIN);
      DCL SORTOUT FILE INPUT RECORD,
          CHARS CHAR(80) VAR;
          ON ENDFILE(SORTOUT) GOTO ENDP;
          L: READ FILE(SORTOUT) INTO (CHARS);
              PUT SKIP EDIT (CHARS) (A);
              GOTO L;
```

```
          ENDP: END P;
/*
//STEP4 EXEC PGM=*.STEP3.LKED.SYSLMOD
//SYSPRINT DD SYSOUT=A
//SORTOUT DD DSNAME=&&TEM,DISP=(OLD,DELETE),UNIT=2314
```

Figure 14-4. PL/I Program Invoking IHESRTA

ENTRY POINT IHESRTB

Entry point IHESRTB is used for sorting records constructed or updated in a PL/I procedure and placing the sorted records in a data set. The format of the CALL statement is:

CALL IHESRTB(argument$_1$,argument$_2$,argument$_3$, argument$_4$,argument$_5$);

where:

argument$_1$-$_4$ = as for IHESRTA

argument$_5$ = entry name of the PL/I procedure supplying the records to the sort program.

The PL/I records are passed to an entry point in the sort program called a user exit E15. (A user exit is a point in the executable code of the sort program at which control can be received from or passed to a user program.)

A PL/I procedure invoked from user exit E15 uses a RETURN statement to pass to the sort program a character-string representation of the record to be sorted. If the record is not in character-string form, it must be defined on a character string and then passed. This may lead to difficulties in the PL/I program. The language rules specify that the attributes of the defined and the base items must match exactly except for their lengths, and that string overlay defining on an aggregate parameter is not permitted. The implementation of the PL/I (F) compiler permits the use of differing attributes and of this type of overlay defining, and produces error(E) diagnostics when these situations occur. Successful link-editing and execution are possible, provided the condition codes in the appropriate EXEC statements allow the step concerned to be executed.

Return Codes from PL/I to Sort

In addition to the return code supplied by the sort program to the PL/I program, the PL/I program must pass a return code to the sort program to indicate whether there are any more records to be passed for sorting. This return code is set by one of the following statements:

```
CALL IHESARC(n); (single-task programs)
CALL IHETSAC(n); (multitasking programs)
```

where n has the values:

```
 8  No more records will be passed
12  Sort the next record to be passed
```

If the CALL IHETSAC(n) statement is to be used, the TASK option must be specified in the main PROCEDURE statement.

IHESARC or IHETSAC must be declared as an entry of precision (31,0), for example:

```
DCL IHESARC ENTRY(FIXED BINARY(31,0));
```

Example

An example of a PL/I program that uses IHESRTB is given in Figure 14-5. This program sorts records similar to those in the example in Figure 14-4. The PL/I procedure E15A is invoked from the sort program user exit E15. This procedure returns a character string that is inserted by the sort program into the sort. The sorted records are transmitted by the sort program to a temporary data set defined by the DD statement SORTOUT, from which they are retrieved and printed by the PL/I program in job steps STEP3 and STEP4.

```
//R20B JOB
//STEP1 EXEC PL1LFCL
//PL1L.SYSIN DD *
   /* PL/I PROGRAMMING EXAMPLE USING IHESRTB */

   SORTB: PROC OPTIONS (MAIN);

   /* DECLARE SORT PROGRAM ENTRY AND EXIT POINTS */

      DCL IHESRTB ENTRY(CHAR(35),CHAR(27),FIXED(31,0),
                FIXED BIN(31,0),ENTRY),
          IHESARC ENTRY(FIXED BIN(31,0)),
          E15A ENTRY RETURNS(CHAR(80)),
          RETURN_CODE FIXED BIN(31,0);

   /* INVOKE THE SORT PROGRAM */

      CALL IHESRTB (' SORT FIELDS=(75,6,CH,A,16,6,CH,A) ',
                    ' RECORD TYPE=F,LENGTH=(80) ',
                    25000, /* MAIN STORAGE FOR SORT PROGRAM */
                    RETURN_CODE,E15A);

   /* TEST RETURN CODE */

     IF RETURN_CODE = 16 THEN PUT SKIP EDIT ('SORT FAILED')(A);
     ELSE IF RETURN_CODE = 0 THEN PUT SKIP EDIT ('SORT COMPLETE')(A);
          ELSE PUT SKIP EDIT ('INVALID SORT RETURN CODE.  CODE=',
                              RETURN_CODE)(A);

   E15A:    /* THIS PROCEDURE OBTAINS RECORDS FROM THE INPUT STREAM */
            /*  AND CHECKS FOR NUMERIC OR ALPHABETIC CODES BEFORE    */
            /* PASSING ONLY THOSE WITH ALPHABETIC CODES TO THE SORT  */
            /* PROGRAM.  RECORDS WITH NUMERIC CODES ARE LISTED.      */

            PROC RETURNS(CHAR(80));
                 DCL SYSIN FILE RECORD INPUT;
                   ON ENDFILE(SYSIN) BEGIN;
                   PUT SKIP(3) EDIT ('END OF SORT PROGRAM INPUT')(A);
                   CALL IHESARC(8); /* SIGNAL END OF SORT INPUT */
                   GOTO ENDE15;
                   END;
                 DCL INFIELD CHAR(80), FIELD1 CHAR(6) DEF INFIELD
                 POS(75);
          NEXT:  READ FILE (SYSIN) INTO (INFIELD);
                 IF FIELD1 > 'ZZZZZZ' THEN DO;
                   PUT SKIP EDIT (INFIELD) (A);
                   GOTO NEXT;
                   END;
                 CALL IHESARC(12); /* INPUT TO SORT CONTINUES */
                 RETURN (INFIELD);
          ENDE15: END E15A;


      END SORTB;
/*
//STEP2 EXEC PGM=*.STEP1.LKED.SYSLMOD
//SYSOUT DD SYSOUT=A
//SORTOUT DD DSNAME=&&TEM,DISP=(NEW,PASS),
// SPACE=(TRK,(1,1)),UNIT=2314,DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSPRINT DD SYSOUT=A
//SORTLIB DD DISP=SHR,DSN=SYS1.SORTLIB
//SORTWK01 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK02 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK03 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK04 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK05 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK06 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SYSIN DD *
```

```
               AAAAAA                                                  AAAAAA
               ZZZZZZ                                                  ZZZZZZ
               444444                                                  444444
               CCCCCC                                                  ZZZZZZ
               XXXXXX                                                  999999
               CCCCCC                                                  ZZZZZP
               CCCCCC                                                  888888
               CCCCCC                                                  AAAAAA
               CCCCCC                                                  ZZZZZZ
               CCCCCC                                                  ZZZZZA
               333333                                                  ZZZZZZ
               VVVVVV                                                  AAAAAA
               EEEEEE                                                  AAAAAA
               333333                                                  ZZZZZZ
/*
//STEP3 EXEC PL1LFCL
//PL1L.DUMB DD DSNAME=&&GOSET(GO),DISP=(OLD,DELETE)
//PL1L.SYSIN DD *
  /* PL/I ROUTINE TO PRINT OUTPUT FROM SORT PROGRAM EXAMPLES */
  P: PROC OPTIONS(MAIN);
      DCL SORTOUT FILE INPUT RECORD,
          CHARS CHAR(80) VAR;
          ON ENDFILE(SORTOUT) GOTO ENDP;
          L: READ FILE(SORTOUT) INTO (CHARS);
            PUT SKIP EDIT (CHARS) (A);
            GOTO L;
          ENDP: END P;
/*
//STEP4 EXEC PGM=*.STEP3.LKED.SYSLMOD
//SYSPRINT DD SYSOUT=A
//SORTOUT DD DSNAME=&&TEM,DISP=(OLD,DELETE),UNIT=2314
```

Figure 14-5. PL/I Program Invoking IHESRTB

ENTRY POINT IHESRTC

Entry point IHESRTC is used for sorting records from a data set and then passing them one-by-one to a PL/I procedure. The format of the CALL statement is:

CALL IHESRTC(argument$_1$,argument$_2$,argument$_3$,
argument$_4$,argument$_6$);

where:

argument$_{1-4}$ = as for IHESRTA

argument$_6$ = entry name of the PL/I procedure to which the sorted records are to be passed.

When IHESRTC is used, each record that appears in the sorted output is passed to user exit E35. The PL/I procedure associated with this user exit is invoked for each record that it receives as a parameter.

The records passed by the sort program must be in a character-string form. If this form is not the one required by the PL/I program, then the PL/I record must be defined on a character string. The same difficulties can be expected here as for IHESRTB.

Return Codes from PL/I to Sort

A return code may be passed by the PL/I procedure to the sort program, using the CALL IHESARC(n) or CALL IHETSAC(n) statement. The return code values are:

4 the record passed has been accepted, pass the next record

8 stop passing records, even if there are still more to come.

IHESARC and IHETSAC must be declared as an entry of precision (31,0).

If no return code is passed the sort program continues to pass records until all have been passed.

An example of a PL/I program that uses IHESRTC is given in Figure 14-6. This program sorts records similar to those in the previous examples. The PL/I procedure E35A is invoked from the sort program user

```
//R20C JOB
//STEP1 EXEC PL1LFCL
//PL1L.SYSIN DD *
   /* PL/I PROGRAMMING EXAMPLE USING IHESRTC */


   SORTC:   PROC OPTIONS (MAIN);


   /* DECLARE SORT PROGRAM ENTRY AND EXIT POINTS */

      DCL IHESRTC ENTRY(CHAR(35),CHAR(27),FIXED BIN(31,0),
                FIXED BIN(31,0),ENTRY),
          IHESARC ENTRY(FIXED BIN(31,0)),
          E35A ENTRY,
          RETURN_CODE FIXED BIN(31,0);


   /* INVOKE THE SORT PROGRAM */

      CALL IHESRTC (' SORT FIELDS=(75,6,CH,A,16,6,CH,A) ',
                  ' RECORD TYPE=F,LENGTH=(80) ',
                  25000, /* MAIN STORAGE FOR SORT PROGRAM */
                  RETURN_CODE, E35A);

   /* TEST RETURN CODE */

     IF RETURN_CODE = 16 THEN PUT SKIP EDIT ('SORT FAILED')(A);
     ELSE IF RETURN_CODE = 0 THEN PUT SKIP EDIT ('SORT COMPLETE')( A);
          ESLE PUT SKIP EDIT ('INVALID SORT RETURN CODE.CODE=',
                              RETURN_CODE)(A);



     E35A:   /* THIS PROCEDURE OBTAINS SORTED RECORDS FROM THE  */
             /* SORT PROGRAM AND LISTS THEM.  DUPLICATE RECORDS  */
             /* ARE IGNORED.

         PROC (INREC);

            /* PRINT HEADING FOR SORTED OUTPUT ON SYSPRINT */

               DCL I STATIC INIT(0);
               IF I = 0 THEN DO;
                    PUT SKIP EDIT ('OUTPUT FROM E35 SUBROUTINE')(A);
                    I=1;
                    END;

            /* PROCESS SORTED RECORDS */
            DCL INREC CHAR(80),
                PREVREC CHAR(80) STATIC INIT(' ');
            IF INREC=PREVREC THEN GOTO NEXT; /* IGNORE THIS RECORD */
            ELSE DO;
            PREVREC=INREC; /* STORE CURRENT RECORD */
                      PUT SKIP EDIT (INREC) (A);
                      END;
     NEXT: CALL IHESARC(4); /* REQUEST NEXT RECORD FROM SORT */
           END E35A;
      END SORTC;
/*
//STEP2 EXEC PGM=*.STEP1.LKED.SYSLMOD
//SYSOUT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SORTLIB DD DISP=SHR,DSN=SYS1.SORTLIB
//SORTWK01 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK02 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK03 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK04 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
```

```
//SORTWK05 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK06 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTIN DD *
               AAAAAA                                                AAAAAA
               ZZZZZZ                                                ZZZZZZ
               444444                                                444444
               CCCCCC                                                ZZZZZZ
               XXXXXX                                                999999
               CCCCCC                                                ZZZZZZ
               CCCCCC                                                AAAAAA
               CCCCCC                                                AAAAAA
               CCCCCC                                                ZZZZZP
               CCCCCC                                                ZZZZZZ
               333333                                                ZZZZZZ
               VVVVVV                                                AAAAAA
               EEEEEE                                                AAAAAA
               333333                                                ZZZZZZ
/*
```

Figure 14-6. PL/I Program Invoking IHESRTC

exit E35. This procedure receives a character-string representing a sorted record from the sort program. The input to the sort program is in the input stream.

ENTRY POINT IHESRTD

Entry point IHESRTD is used for sorting records constructed or updated by a PL/I function and then passing them one-by-one to another procedure. The format of the CALL statement is:

```
CALL IHESRTD(argument1,argument2,argument3,
             argument4,argument5,argument6);
```

where:

$arguments_{1-4}$ = as for IHESRTA

$argument_5$ = entry name of the PL/I procedure supplying the records to the sort program

$argument_6$ = entry name of the PL/I procedure to which the sorted records are to be passed

An example of a PL/I program that uses IHESRTD is given in Figure 14-7. This program sorts records similar to those in the previous examples. The PL/I procedure E15A is invoked from the sort program user exit E15. This procedure return a character string that is inserted by the sort program into the sort. At the end of the sort, the sorted records are associated with a character string parameter in the PL/I procedure E35A, which is invoked from the sort user exit E35 for each record emerging from the sort.

```
//R20D JOB
//STEP1 EXEC PL1LFCL
//PL1L.SYSIN DD *
   /* PL/I PROGRAMMING EXAMPLE USING IHESRTD */

   SORTD:   PROC OPTIONS (MAIN);

   /* DECLARE SORT PROGRAM ENTRY AND EXIT POINTS */

      DCL  IHESRTD ENTRY(CHAR(35),CHAR(27),FIXED BIN(31,0),
                  FIXED BIN(31,0),ENTRY,ENTRY),
           IHESARC ENTRY(FIXED BIN(31,0)),
           E15 ENTRY RETURNS(CHAR(80)),
           E35 ENTRY,
           RETURN_CODE FIXED BIN(31,0);

   /* INVOKE THE SORT PROGRAM */

      CALL IHESRTD (' SORT FIELD=(75,6,CH,A,16,6,CH,A) ',
                    ' RECORD TYPE=F,LENGTH=(80) ',
                    25000, /* MAIN STORAGE FOR SORT PROGRAM */
                    RETURN_CODE, E15A,E35A);

   /* TEST RETURN CODE */

     IF RETURN_CODE = 16 THEN PUT SKIP EDIT ('SORT FAILED')(A);
     ELSE IF RETURN_CODE = 0 THEN PUT SKIP EDIT ('SORT COMPLETE')(A);
          ELSE PUT SKIP EDIT ('INVALID SORT RETURN CODE.CODE=',
                              RETURN_CODE)(A);

  E15A:   /* THIS PROCEDURE OBTAINS RECORDS FROM THE INPUT STREAM */
         /* AND CHECKS FOR NUMERIC OR ALPHABETIC CODES BEFORE    */
         /* PASSING ONLY THOSE WITH ALPHABETIC CODES TO THE SORT */
         /* PROGRAM.  RECORDS WITH NUMERIC CODES ARE LISTED.     */

         PROC RETURNS(CHAR(80));
              DCL SYSIN FILE RECORD INPUT;
              ON ENDFILE(SYSIN) BEGIN;
                PUT SKIP(3) EDIT ('END OF SORT PROGRAM INPUT.',
                    'SORTED OUTPUT SHOULD FOLLOW')(A);
                CALL IHESARC(8); /* SIGNAL END OF SORT INPUT */
                GOTO ENDE15;
                END;
         DCL INFIELD CHAR(80), FIELD1 CHAR(6) DEF INFIELD
             POS(75);
      NEXT:  READ FILE (SYSIN) INTO (INFIELD);
             IF FIELD1 > 'ZZZZZZ' THEN DO;
               PUT SKIP EDIT (INFIELD)(A);
               GOTO NEXT;
               END;
             CALL IHESARC(12); /* INPUT TO SORT CONTINUES */
             RETURN (INFIELD);
    ENDE15: END E15A;
  E35A:   /* THIS PROCEDURE OBTAINS SORTED RECORDS FROM THE  */
         /* SORT PROGRAM AND LISTS THEM.  DUPLICATE RECORDS */
         /* ARE IGNORED.                                    */

     PROC (INREC);

         /* PRINT HEADING FOR SORTED OUTPUT ON SYSPRINT */

            DCL I STATIC INIT(0);
            IF I = 0 THEN DO;
                 PUT SKIP EDIT ('OUTPUT FROM E35 SUBROUTINE')(A);
                 I=1;
            END;

         /* PROCESS SORTED RECORDS */
```

```
          DCL INREC CHAR(80),
              PREVREC CHAR(80) STATIC INIT(' ');
              IF INREC=PREVREC THEN GOTO NEXT; /* IGNORE THIS RECORD */
               ELSE DO;
                    PREVREC=INREC; /* STORE CURRENT RECORD */
                    PUT SKIP EDIT (INREC) (A);
                    END;
     NEXT:  CALL IHESARC(4); /* REQUEST NEXT RECORD FROM SORT */
            RETURN;
            END E35A;
      END SORTD;
/*
//STEP2 EXEC PGM=*.STEP1.LKED.SYSLMOD
//SYSOUT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SORTLIB DD DISP=SHR,DSN=SYS1.SORTLIB
//SORTWK01 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK02 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK03 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK04 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK05 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK06 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SYSIN DD *
               ZZZZZZ                                                  AAAAAA
               ZZZZZZ                                                  ZZZZZZ
               444444                                                  444444
               CCCCCC                                                  ZZZZZZ
               XXXXXX                                                  999999
               CCCCCC                                                  ZZZZZZ
               CCCCCC                                                  888888
               CCCCCC                                                  AAAAAA
               CCCCCC                                                  ZZZZZP
               333333                                                  ZZZZZZ
               VVVVVV                                                  AAAAAA
               EEEEEE                                                  AAAAAA
               333333                                                  ZZZZZZ
/*
```

Figure 14-7. PL/I Program Invoking IHESRTD

SORTING VARIABLE-LENGTH RECORDS

When you wish to use the PL/I sorting facilities to sort variable-length records, you should note the following points:

1. The portion of a variable-length record that contains the control field or fields on which the sort is to be performed must be present and of the same length for every record to be sorted. A sort cannot be performed on control fields whose length or position within a record is liable to alter. Thus the control fields would be expected within the minimum length given for the records in the RECORD control statement.

2. The length of each record is recorded in the first four bytes of the record. Provision for this length field should be made when you specify the sort control fields in the SORT control statement.

3. Varying-length strings passed from a PL/I user exit E15 procedure will have the length field added to the record automatically; the length will be the current length of the character string plus four bytes for the field itself. The same applies if fixed-length strings of different lengths are returned from the E15 procedure.

4. The four-byte length field is removed from variable-length records passed to a PL/I user exit E35 routine.

An example of a PL/I program that uses IHESRTA to sort variable-length records is given in Figure 14-8. This example includes a PL/I program to create a data set of variable-length records from data items obtained from the input stream. The sort is performed on alphanumeric data in the first six bytes following the length field in each record. A third PL/I program retrieves the sorted variable-length records from a temporary data set and lists them. Note that the maximum record length includes four bytes for the length field,

and corresponds to the maximum length given in the LRECL subparameter.

USE OF PL/I SORT IN A MULTITASKING ENVIRONMENT

When the sort program is invoked from different PL/I tasks, so that two or more sorting operations are to be performed asynchronously by separate subtasks, the following should be noted if the sort program diagnostic messages are to be printed on the line printer:

If the DD statement for the SYSOUT data set contains SYSOUT=A in the operand field, some sort program messages may be overwritten in the data management buffers and therefore not printed. Further, the program in some cases will terminate abnormally or go into a wait state. These problems are caused by the inability to modify the ddname for the SYSOUT data set in the additional tasks that use the sort program; they give rise to synchronization conflicts within data management.

This problem does not apply if, when the system is generated, the sort program messages are specified to be printed on the console.

```
//R20V JOB
//STEP1 EXEC PL1LFCLG
//PL1L.SYSIN DD *
   VAR2:  PROC OPTIONS(MAIN);
          ON ENDFILE(SYSIN) GOTO END;
          DCL OUT FILE RECORD OUTPUT,
              OUTREC CHAR (80) VAR;
     NEXT: GET LIST (OUTREC);
           PUT SKIP EDIT (OUTREC) (A);
           WRITE FILE (OUT) FROM (OUTREC);
           GOTO NEXT;
     END:  END VAR2;
/*
//GO.OUT DD DSNAME=&&TEMP,DISP=(NEW,PASS),SPACE=(TRK,(1,1)),
// DCB=(,RECFM=V,LRECL=84),UNIT=2314
//GO.DUMB DD DSNAME=&&GOSET(GO),DISP=(OLD,DELETE)
//GO.SYSIN DD *
'003329HOOKER S.W. RIVERDALE, SATCHWELL LANE, BACONSFIELD'
'002886BOOKER R.R. ROTORUA, MILKEDGE LANE, TOBLEY'
'003077ROKKER & SON, LITTLETON NURSERIES, SHOLTSPAR'
'059334HOOK E.H. 109 ELMTREE ROAD, GANNET PARK, NORTHAMPTON'
'73872HOME TAVERN, WESTLEIGH'
'000931FOREST, IVER, BUCKS'
/*
//STEP2 EXEC PL1LFCL
//PL1L.SYSIN DD *
   VARY1: PROC OPTIONS(MAIN);
        DCL IHESRTA ENTRY (CHAR(24),CHAR(35),
                   FIXED BIN(31,0),FIXED BIN(31,0)),
            RETURN_CODE FIXED BIN(31,0);
        CALL IHESRTA(' SORT FIELDS=(5,6,CH,A) '
                     ' RECORD TYPE=V,LENGTH=(84,,,20,40) ',
                     25000, /* MAIN STORAGE FOR THE SORT PROGRAM */
                     RETURN_CODE);
            IF RETURN_CODE=0 THEN PUT SKIP EDIT
               ('SORT COMPLETE')(A);
                           ELSE IF RETURN_CODE=16 THEN PUT SKIP EDIT
                            ('SORT FAILED')(A);
                                ELSE PUT SKIP EDIT
                                 ('INVALID SORT RETURN CODE')(A);
   END VARY1;
/*
//STEP3 EXEC PGM=*.STEP2.LKED.SYSLMOD
//SYSOUT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SORTLIB DD DISP=SHR,DSN=SYS1.SORTLIB
//SORTWK01 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK02 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK03 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK04 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK05 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTWK06 DD UNIT=2314,SPACE=(TRK,(60,20),,CONTIG)
//SORTIN DD DSNAME=&&TEMP,DISP=(OLD,DELETE),
//  DCB=(BLKSIZE=88,LRECL=84,RECFM=V)
//SORTOUT DD DSNAME=&&TEM,DISP=(NEW,PASS),
// SPACE=(TRK,(1,1)),UNIT=2314,DCB=(RECFM=V,LRECL=84,BLKSIZE=88)
//STEP4 EXEC PL1LFCL
//PL1L.DUMB DD DSNAME=&&GOSET(GO),DISP=(OLD,DELETE)
//PL1L.SYSIN DD *
  /* PL/I ROUTINE TO PRINT OUTPUT FROM SORT PROGRAM EXAMPLES */
  P:  PROC OPTIONS(MAIN);
      DCL SORTOUT FILE INPUT RECORD,
          CHARS CHAR(80) VAR;
          ON ENDFILE(SORTOUT) GOTO ENDP;
          L: READ FILE(SORTOUT) INTO (CHARS);
             PUT SKIP EDIT (CHARS)(A);
             GOTO L;
```

```
          ENDP: END P;
/*
//STEP5 EXEC PGM=*.STEP4.LKED.SYSLMOD
//SYSPRINT DD SYSOUT=A
//SORTOUT DD DSNAME=&&TEM,DISP=(OLD,DELETE),UNIT=2314
```

Figure 14-8. Using IHESRTA to Sort Variable-length Records

Chapter 15: PL/I and Other Languages

Introduction

This chapter is presented as two main sections. The first is concerned with the subject of data set interchange (i.e., sharing data sets between programs written in different languages); the second is concerned with the more complex subject of linkage (i.e., direct communication) between modules written in different languages but forming one program.

Data Set Interchange

In general, a PL/I program can use data sets produced by programs written in other languages, and can produce data sets that can be used by these other programs. There are some limitations, caused mainly by differences in the way in which PL/I data is handled, in particular the way in which it is mapped and stored, and by the large number of different data types available in PL/I that are not available in the other languages. However, there are many data types and modes of storage in PL/I that have equivalents in other high-level languages. There are also ways in which most of the incompatibilities can be overcome. A previous incompatibility has been removed in the fifth version of the PL/I (F) compiler by the use of halfword storage for data that is FIXED BINARY and has a precision of less than 16.

PL/I-FORTRAN DATA SET INTERCHANGE

The major areas of incompatibility between PL/I and FORTRAN are as follows:

1. PL/I has more data types; therefore some PL/I data types have no equivalent in FORTRAN, notably character-string data.

2. PL/I records can be organized as structures in main storage (with the elements aligned or unaligned). FORTRAN records are built up on output, and the elements will not be aligned unless written in a particular order.

3. PL/I arrays are stored in row major order, while FORTRAN arrays are stored in column major order.

Equivalent Data Types

Figure 15-1 shows FORTRAN-PL/I data equivalents. Note that PL/I numeric character data has no direct equivalent in FORTRAN. However, it is possible to treat a FORTRAN array as a PL/I character string.

Structures and Alignment

The safest course when PL/I structures are to be associated with FORTRAN records is to declare the structures UNALIGNED. However, if the elements of the FORTRAN record have been forced into correct alignment (using techniques such as writing out the variables that form the record in descending order of data type length), the PL/I structure can be declared ALIGNED. Note that unaligned data is less efficient than aligned data, often requiring extra execution time when accessed.

Example:

FORTRAN

```
Main storage: INTEGER*2A
              LOGICAL*1B
              REAL*8C

Output:       WRITE(6) C,A,B
```

PL/I

```
Main storage: DCL 1 R UNALIGNED,
                  2 U FLOAT(16),
                  2 S FIXED BIN,
                  2 T BIT(8);

Input:        READ FILE(FILE6) INTO(R);
```

FORTRAN (Unformatted Record)			PL/I			
					Alignment	
Data Type	Length (bytes)	Alignment[1]	Data Type	Length (bytes)	Aligned	Unaligned
INTEGER*2	2	Byte	FIXED BINARY(15)	2	Halfword	Byte
INTEGER*4	4	Byte	FIXED BINARY(31)	4	Fullword	Byte
REAL*4	4	Byte	REAL FLOAT(short)	4	Fullword	Byte
REAL*8	8	Byte	REAL FLOAT(long)	8	Doubleword	Byte
COMPLEX*8	8	Byte	COMPLEX FLOAT(short)	8	Fullword	Byte
COMPLEX*16	16	Byte	COMPLEX FLOAT(long)	16	Doubleword	Byte
LOGICAL*1	1	Byte	BIT(8)	1	Byte	Bit[2]
LOGICAL*4	4	Byte	BIT(32)	4	Byte	Byte
-	-	-	CHARACTER	any	Byte	Byte

Notes: [1]When allocated, the data types generally have the same alignment as listed under the PL/I equivalents, but after more than one data item has been written out in one record (equivalent to writing a PL/I structure), this alignment may be lost, and therefore all items should be considered to be byte-aligned for input/output.

[2]The fact that the alignment requirement of unaligned bit strings is bit rather than byte does not affect PL/I-FORTRAN data interchange, since the FORTRAN string will always take up an integral number of bytes.

Figure 15-1. FORTRAN-PL/I Data Equivalents

In the above example, S, T, and U are equivalent data types to A, B, and C respectively. As alignment of the elements has been forced in the FORTRAN WRITE statement, R could have been declared ALIGNED in this case.

The record format will be either VS or VBS, since the FORTRAN object-time library automatically uses a spanning technique. The record format specified in the PL/I ENVIRONMENT attribute (or in the DD statement associated with the PL/I file) must match that specified in the DD statement associated with the FORTRAN data set reference number.

Arrays

Arrays of one dimension are stored in the same way in both PL/I and FORTRAN, but arrays of two or more dimensions are stored differently (row major order for PL/I, column major order for FORTRAN). The solution to this problem is to declare the array in one language with subscripts in reverse order to the declaration in the other language. The record will then be correctly associated with the array in either program on transmission.

Example:

FORTRAN

```
REAL A(2,5,7)
```

PL/I

```
DCL A(7,5,2)
```

The disadvantage of this method as it stands is that a reference to the array using the same subscripts in the two different programs will access different elements. A(1,3,6) in FORTRAN would be A(6,3,1) in PL/I; the likelihood of program error may be increased. This problem can be avoided by using iSUB defining in the PL/I program.

Example:

FORTRAN

```
REAL A(2,5,7)
```

PL/I

```
DCL A1(7,5,2),
    A(2,5,7) DEF A1(3SUB,2SUB,1SUB);
```

Now, if the record is transmitted to or from A1 in the PL/I program, a reference to a particular element in the FORTRAN program will be identical to the PL/I reference to that element.

External Representation of Floating-Point Numbers

The preceding discussions are concerned with the internal representation of equivalent data types in FORTRAN and PL/I, and the interchange of record-oriented data sets. The following notes describe an incompatibility between PL/I and FORTRAN in the external representation of floating-point numbers in stream-oriented data sets.

1. For PL/I, the exponent of any floating-point number is indicated by the character "E". For FORTRAN, the exponent of a short-precision floating-point number is indicated by the character "E" and the exponent of a long-precision floating-point number is indicated by the character "D".

2. For PL/I, if the exponent of a floating-point number is positive, it may optionally have a "+" sign. If a sign is not present, the exponent immediately follows the character "E". For example:

0.018737E22

or

0.018737E+22

For FORTRAN, if the exponent of a floating-point number is positive, a blank character rather than a '+' sign may sometimes be present between the "E" or "D" and the exponent. For example:

0.018737D 22

These differences prevent the direct use of PL/I stream-oriented transmission facilities to read or write floating-point numbers that can be interchanged with a FORTRAN program.

PL/I-COBOL DATA SET INTERCHANGE

PL/I has data types and organizations that correspond to most COBOL items, but there are some differences. For example, PL/I data may be either aligned or unaligned, but records produced by a program compiled by a non-ANS COBOL compiler are always aligned (for ANS COBOL, see below). Also, PL/I structure mapping differs from COBOL record mapping in that the PL/I (F) compiler minimizes the amount of unused storage within the structure. This means that corresponding elements may occupy different storage locations relative to the start of the structure, and that the record length may differ from that of the COBOL record.

ANS COBOL data can be SYNCHRONIZED (i.e., aligned) or unsynchronized (i.e., unaligned). COBOL records prior to USASI COBOL were always SYNCHRONIZED, and the fifth version of the PL/I (F) compiler assumes that structures in a file with the COBOL option are SYNCHRONIZED. Note that for ANS COBOL the default is unsynchronized, while for PL/I the default is aligned for all data types except bit-string, character-string, and numeric character.

You can use the COBOL option in the ENVIRONMENT attribute for a PL/I file that is to be associated with a COBOL data set. This indicates that any structure in the data set associated with this file is mapped according to the COBOL algorithm (i.e., starting at a doubleword boundary and, proceeding left to right, aligning elements on the first available correct boundary). If a structure name appears in a READ INTO or WRITE FROM statement for a file with the COBOL option, a temporary COBOL-type structure for transmission to or from the data set is created. Before output (or after input), the elements of the PL/I-mapped structure are assigned to (or from) the temporary. If the PL/I structure and the COBOL record have identical mapping, the data sets will be directly compatible and there will be no need to use the COBOL option. Note that the following restrictions apply to files with the ENVIRONMENT(COBOL) attribute.

1. The file can be used only for READ INTO, WRITE FROM, and REWRITE FROM statements. The file name cannot be passed as an argument.

2. The variable named in the INTO option cannot be used in the on-unit for an ON-condition raised during execution of a READ statement.

3. The EVENT option can be used only if the compiler can determine that the PL/I structure and the COBOL record have identical mapping. (For further details see the PL/I (F) Language Reference Manual.)

Figure 15-2 shows COBOL-PL/I data equivalents.

Linkage with Other Languages

The information so far presented in this chapter has dealt with data set interchange between PL/I and non-PL/I modules run as separate jobs or job steps (i.e., run as separate programs sharing a data set). The rest of the chapter is devoted to direct communication between PL/I modules and non-PL/I modules that are run in the same job step (i.e., forming one program). Because of the different conventions and methods adopted by the different processors, you are advised not to attempt the type of operation discussed below without some knowledge of the execution-time environment; you will require access to the publication IBM System/360 Operating System: PL/I Subroutine Library Program Logic Manual, Form Y28-6801.

This section is in two parts and consists of a brief discussion of the PL/I (F) environment, pointing out the particular areas that you should be familiar with in order to carry out interlanguage communications, followed by a discussion of the communication problems and suggested solutions.

Note: This section devotes itself mainly to a single tasking system. Therefore, where there is no mention of multitasking , single tasking should be assumed.

PL/I (F) ENVIRONMENT AND COMMUNICATIONS

The following paragraphs deal with the way in which the PL/I (F) environment is set up and briefly describe the more important control blocks used for communications. Note that this section provides only a summary of the relevant information, which is given in detail in the PL/I Subroutine Library Program Logic Manual. The purpose of the section is to provide an overall picture of the areas that you may need to consider.

COBOL				PL/I			
		Alignment				Alignment	
Data Type	Length (bytes)	Synch. (aligned)	Unsynch. (unaligned)	Data Type	Length (bytes)	Aligned	Unaligned
COMPUTATIONAL[1] dec. length:							
1-4	2	Halfword	Byte	FIXED BINARY (halfword integer)	2	Halfword	Byte
5-9	4	Fullword	Byte	FIXED BINARY (fullword integer)	4	Fullword	Byte
10-18	8	Fullword	Byte	No equivalent	-	-	-
COMPUTATIONAL-1	4	Fullword	Byte	REAL FLOAT(short)	4	Fullword	Byte
COMPUTATIONAL-2	8	Doubleword	Byte	REAL FLOAT(long)	8	Doubleword	Byte
COMPUTATIONAL-3	1[2]	Byte	Byte	FIXED DEC	1[2]	Byte	Byte
DISPLAY	any	Byte	Byte	CHARACTER	any	Byte	Byte

Notes: [1]Decimal length is equal to the number of 9s in the picture.

[2]The length of 1 byte applies to the smallest fixed decimal value (i.e., 1 digit). For other values, the length is given by CEIL((number of digits + 1)/2) bytes.

Figure 15-2. COBOL-PL/I Data Equivalents

The PL/I (F) Environment

The (F) compiler is designed so that resulting object programs, whether multitasking in MVT or single tasking in PCP or MFT, can use storage efficiently by obtaining storage for each block when required and releasing it when no longer needed. This is achieved by the use of library modules, called by housekeeping routines for each block. These housekeeping routines (prologues and epilogues) perform various other functions (such as saving the environment of the invoking block), described in more detail later.

Because of the need to be able to operate in a multiprogramming system, all PL/I library modules are reentrant. (Your output from the compiler will also be reentrant if you use the static storage class only for read-only data and specify the REENTRANT option in the PROCEDURE statement.)

As well as the block housekeeping routines, the procedure with the OPTIONS(MAIN) option needs routines to set up the PL/I environment. These functions are performed by the library module IHESAP (or IHETSA in a multitasking program) only once for any given execution of a program. They consist of the following:

1. Obtaining the length of the pseudo-register vector (PRV) from the linkage editor. (The PRV is an area of task-oriented storage that contains a number of pseudo-registers; these hold information such as the invocation count).

2. Allocating the PRV variable data area (PRV VDA). This area contains the PRV plus primary library workspace (LWS). General register 12 is set to point to the PRV.

3. Initializing the standard pseudo-registers and the LWS.

4. Issuing a SPIE macro instruction to allow the library module IHEERR to obtain control in case of a program interrupt.

5. Issuing a STAE macro instruction to intercept abnormal termination by the system.

6. Chaining the PRV VDA back to the external save area.

These functions are the result of the initial control from the operating system being passed to the library module IHENTRY. IHENTRY then selects the proper IHESAP entry point to utilize the specific environment required. (IHESAP ultimately passes control to the main procedure via the control section IHEMAIN.) Since this environment is established only once, all procedures without the OPTIONS(MAIN) option assume that the environment has been previously established and has not been disturbed. Note that the technique of using IHENTRY enables the generated code for the main procedure to be similar to that for any other external procedure.

Invoking a PL/I Main Procedure

Initial Entry to Procedures with the MAIN Option: In order to achieve the proper initialization of PL/I programs, the primary entry is always to a compiler generated control section, IHENTRY. This calls the appropriate PL/I library initialization routine (IHESAP or IHETSA); the choice of module depends on the OPT parameter and whether the main procedure has the TASK option or not. The routine selected provides the PRV and library workspace, issues SPIE and STAE macros and then transfers control to the address contained in a control section named IHEMAIN. This address constant is produced by the compiler for each external procedure with the MAIN option. The situation is illustrated in Figure 15-3.

If more than one module has the MAIN option, the linkage editor will accept the first appearance of the control section IHEMAIN in its input stream and ignore the rest. Thus, if more than one main procedure is to be called, IHEMAIN must be dynamically altered to hold the address of the required PL/I main procedure before calling IHENTRY.

If an argument is to be passed to the PL/I program, IHESAP or IHETSA must be entered at the appropriate entry point. This entry point can be called directly, or can be placed in IHENTRY and thus entered by the call to IHENTRY.

For non-tasking the entry points in IHESAP have the following functions:

IHESAPA
parameter from EXEC statement OPT=0

IHESAPB
normal parameter conventions OPT=0

IHESAPC
parameter from EXEC statement OPT=1 or 2

IHESAPD
normal parameter conventions OPT=1 or 2

For tasking the entry points in IHETSA have the following functions:

IHETSAP
parameter from EXEC statement.

IHETSAA
normal parameter conventions.

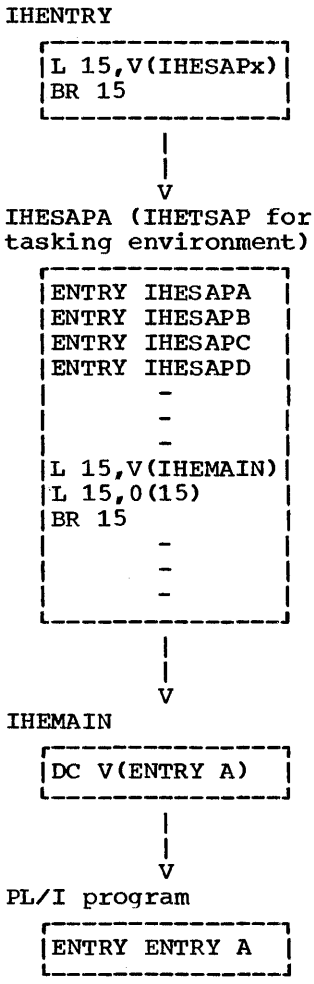
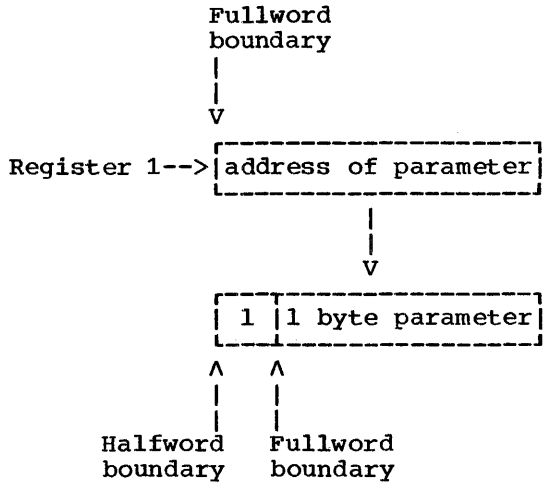
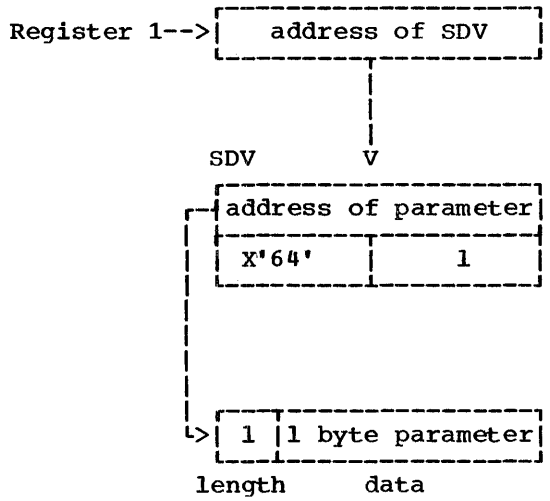


Figure 15-3. Initial Entry to Procedures with the MAIN Option.

Passing Arguments to a Main Procedure: If the entry points IHESAPA, IHESAPC or IHETSAP are called the procedure will expect a single character string argument from the PARM field of the EXEC statement. Register 1 should be pointing at the address of the parameter, as shown below.



The maximum length of the argument is 100 bytes and should be declared as CHAR(100) VARYING in the main procedure. The housekeeping module IHESAP/IHETSA builds a string dope vector to describe the argument as a varying character string so that on entry to the main procedure the set up will be:



Example:

PL/I PROGRAM

```
MYPROC: PROC (PARAM) OPTIONS(MAIN);
        DCL PARAM CHAR (100)VARYING;
        -
        -
        -
        END MYPROC;
```

ASSEMBLER LANGUAGE CALLING PROGRAM

```
L     1,PARM
L     15,SAPA
BALR  14,15
      -
      -
      -
SAPA  DC  V(IHESAPA)
PARM  DC  A (ADDR)
      DS  H
ADDR  DC  AL2(L'LIST)
LIST  DC  C 'THIS IS PARAMETER'
      -
      -
      -
      END
```

If entry points IHESAPB, IHESAPD or IHETSAA are called the procedure may or may not be expecting an argument list, dependant on whether one was specified in the procedure statement. Register 1 is not examined by the housekeeping module IHESAP/IHETSA, but simply passed on to the main procedure. If no parameters have been declared in the main procedure Register 1 will be ignored, but if parameters are declared Register 1 will be expected to point to a parameter list. The parameters must be in a form acceptable to PL/I.

Note: It is dangerous to assign to the parameter of the EXEC statement. (If length is greater than the current length job may fail).

As an example, consider passing three fixed length character-string arguments to the main procedure.

Example:

PL/I PROGRAM

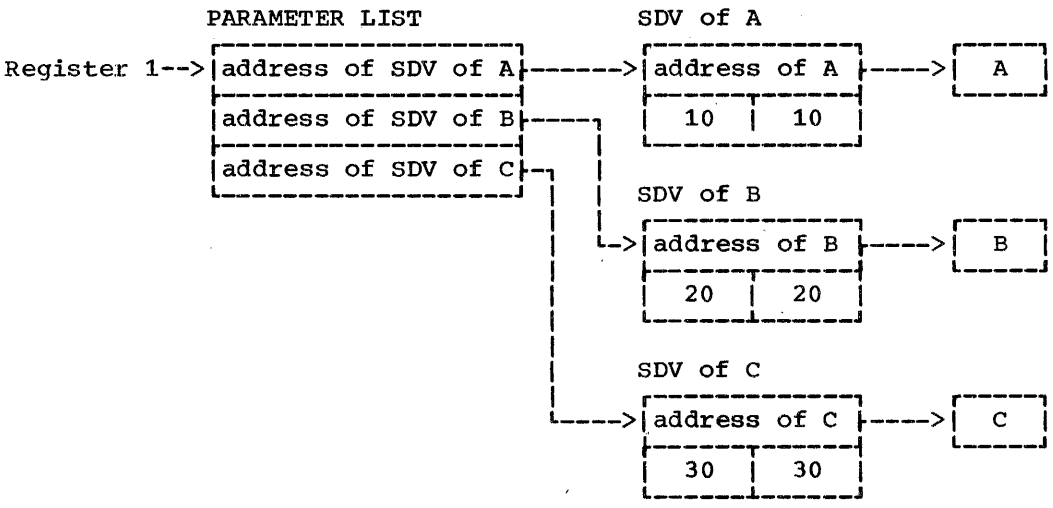
```
MYPROC: PROC (A,B,C) OPTIONS(MAIN);
        DCL  A  CHAR (10);
        DCL  B  CHAR (20);
        DCL  C  CHAR (30);
         -
         -
         -
        END MYPROC;
```

Register 1 will now be set up as follows:

ASSEMBLER LANGUAGE CALLING PROGRAM

```
L     1,PARM
L     15,SAPB
BALR  14,15
      -
      -
      -
SAPB  DC  V(IHESAPB)
PARM  DC  A(SDVA)
      DC  A(SDVB)
      DC  X'80'
      DC  AL3(SDVC)
SDVA  DC  A(PARMA)
      DC  2AL2(L'PARMA)
PARMA DC  CL10'PARAMETERA'
SDVB  DC  A(PARMB)
      DC  2AL2(L'PARMB)
PARMB DC  CL20'PARAMETERB'
SDVC  DC  A(PARMC)
      DC  2AL2(L'PARMC)
PARMC DC  CL30'PARAMETERC'
      -
      -
      -
      END
```

The procedure prologue for each procedure:



1. Sets general register 11 to point to the static internal control section.

2. Allocates a dynamic storage area (DSA) for the procedure. The length is 100 bytes plus the lengths of the automatic variables, parameter lists, and dope vectors for that procedure. The 100 bytes contains a register save area plus interrupt and environment information and the field used to provide statement numbers in object-time error messages.

3. Chains the DSA back to the original DSA or VDA. Sets flags indicating DSA characteristics. Stores the address of the DSA in the PR for that block. The pseudo-register name is one of those generated by extending the procedure name to the left with asterisks if necessary to make seven characters and then appending, in order, a single character from B through Z and the special characters, corresponding to the order of the blocks in the compilation.

4. Initializes any automatic variables having the INITIAL attribute. Initialize all dope vectors for automatic strings, structures, and arrays.

5. Obtains secondary dynamic storage for variables that depend on the results of step 4. These areas are called variable data areas (VDA). VDAs may also be required for temporary work space.

The procedure epilogue is initiated by a call to the library routine IHESAF (an entry point in IHESAP). This routine frees the current DSA and any associated VDAs. A request to free the DSA of a MAIN procedure results in the raising of the FINISH condition, closing all files still open, and freeing all automatic storage for the program. Control is passed from IHESAF via general register 14, which, for MAIN procedures is a transfer to the calling program (usually the operating-system supervisor). For other procedures, return is to the statement following the call to the procedure being terminated, except in a shared library system, when there is another level of control between the PL/I environment and the operating system.

Storage Organization and Control Blocks

Detail formats and field descriptions of the PL/I control blocks are contained in the PL/I Subroutine Library Program Logic Manual. This section briefly discusses how the blocks are used and how they are related to the system and each other. No attempt is made to cover all control blocks, but only those that a programmer might need to refer to. None of these control blocks can be accessed usefully in FORTRAN or COBOL, but an assembler language subroutine, or an assembler language routine that was acting as an interface between PL/I and FORTRAN or COBOL, could make use of their contents. The control blocks discussed are:

1. Library and data generated control blocks
   - String dope vector (SDV)
   - Array dope vector (ADV)
   - String array dope vector (SADV)
   - Structure dope vector
2. Input/output control blocks
   - Declare control block (DCLCB)
   - Open control block (OCB)
   - File control block (FCB)
   - Input/output control block (IOCB)
   - Record dope vector (RDV)
   - Dope vector descriptor (DVD)
3. Storage management control block
   - Dynamic storage area (DSA)

String Dope Vector (SDV): The string dope vector is primarily necessary to support such features as varying-length strings, adjustable strings, and string arguments. For example, the use of a dope vector allows the SUBSTR function to operate without duplicating the actual data that makes up the substring. The result of this function is simply an additional dope vector that describes part of the original string. Since this additional dope vector exists only during the life of the statement in which it occurs, it is created in library workspace.

Array Dope Vector (ADV): The array dope vector allows the bounds of the array to be dynamically defined by the values of expressions evaluated by the program, and allows arrays to be passed as arguments; it also provides the reference for the SUBSCRIPTRANGE condition.

The main contents of the array dope vector are the virtual origin, the

multipliers, and the bounds of the array. The virtual origin (i.e., the location that the element with all zero subscripts has or would have if it existed) and the multipliers are used, together with the appropriate subscripts, in the address calculation for a given element, thus:

$$\text{address} = \text{virtual origin} + \sum_{i=1}^{n} S_i * M_i$$

where n = number of dimensions
S_i = value of ith subscript
M_i = value of ith multiplier

From the viewpoint of dope vector construction, in contrast to dope vector usage, it is best to think of the virtual origin as a location relative to the start of the actual array. You can calculate the necessary virtual origin as follows:

$$VO = (\text{first element address}) - \sum_{i=1}^{n} M_i * LB_i$$

where LB_i = LBOUND(array,i)

The multipliers are calculated as follows:

$$M = l * \prod_{r=i+1}^{n} (HB_r - LB_r + 1)$$

where l = distance between the starts of two consecutive single elements (i.e., generally, the length of a single element)

HB_r = HBOUND(array,r)

LB_r = LBOUND(array,r)

For example, take an array of three dimensions (10,5,6), with elements 5 bytes long, the first of which is located at 12288.

$M_1 = 5 * 5 * 6 = 150$
$M_2 = 5 * 6 = 30$
$M_3 = 5$

and
virtual origin = 12288 - (150 + 30 + 5)
= 12103

The dope vector for based arrays is slightly different from that for nonbased arrays. The virtual origin, instead of containing an actual address, contains the offset of the virtual origin from the first element declared. It can be calculated from the above formula, but setting the address of the first element to zero since it is not known. Because the virtual origin is not necessarily part of the array itself, this offset could be either positive or negative. It is a 3-byte field, and this must be considered when the virtual origin is loaded into a register for address arithmetic. For all except bit-string arrays, the first eight bits in the register will be zero after the virtual origin has been loaded, and will thus represent a positive number even though the offset might be negative.

String Array Dope Vector (SADV): The string array dope vector takes one of two forms, depending on whether the strings are varying or fixed-length. If the strings are fixed, the SADV is an ADV with two halfwords appended, each of which contains the length of the strings in the array. If the strings are varying, the format of the SADV is the same, but certain meanings are different. To determine which format SADV is being used, test the 'current length' field in the SADV. If it is nonzero, the strings are fixed-length, and normal ADV processing can be used to access the string. If the 'current length' field is zero, normal ADV processing will point to secondary dope vectors for each of the varying strings and each element would then be processed as an element variable. This is necessary because each element in the array could have a different current length.

Structure Dope Vector: The structure dope vector is a concatenation of the dope vectors or addresses of all the elements in the structure. If an element has no dope vector (i.e., it is an element variable of arithmetic type), a single fullword address constant points to the element.

Declare Control Block (DCLCB): The DCLCB is the primary compiler-generated control block for input/output operations. It is a read-only block and contains only the information declared for the file (or implied by the declared attributes). Its address is constant throughout the execution of the program and it can be used as the source of all information about the file by any module requiring the information.

A pseudo-register is used for communication with other related control blocks; the DCLCB contains the offset of the file pseudo-register in the PRV. Another pseudo-register, IHEQFOP is used to identify all files open for the program.

Open Control Block (OCB): When a file is opened, the parameter list passed to the PL/I open interface is called the open control block. This block combined with the DCLCB provides the PL/I attributes for opening the file. The result of the opening process is the file control block

(FCB) and the input/output control blocks (IOCBs).

File Control Block (FCB): The FCB contains the PL/I information about the current opening of the file, plus the DCB associated with the file.

Input/Output Control Block (IOCB): The main purpose of the IOCB is to fill the operating system requirement for data event control blocks (DECBs) used by the BSAM and BDAM/BISAM interfaces. The number of IOCBs created varies with the data set organization and access technique.

For BSAM, the number of IOCBs generated is equal to the NCP value specified in the DD statement for the data set or the ENVIRONMENT file attribute. Unless the EVENT option is used, only one IOCB is required. If the EVENT option is used, one IOCB should be created for each input/output operation that could be outstanding at a given point in time. An attempt to use more IOCBs than requested at open time will raise the ERROR condition.

For BDAM and BISAM, only one IOCB is created at open time, and others are created when needed. The IOCB also contains the hidden buffer area if one is required for BSAM or BDAM.

Record Dope Vector (RDV): Whenever a record-oriented input/output operation is requested, one of two control blocks will be passed to the PL/I library. If the variable involved is a varying character string, the parameter is an SDV; for all other variables, an RDV is passed to the PL/I library. Its contents depend on the type of operation, but, basically, it contains the address of the input/output area for READ or WRITE operations, or the buffer address after a LOCATE statement. It also contains the aggregate length of the specified variable.

Dope Vector Descriptor (DVD): The compiler generates a DVD for use when an aggregate has to be mapped at execution time rather than compile time. This occurs primarily for aggregates specified with adjustable bounds.

Dynamic Storage Area (DSA): If you wish to incorporate a non-PL/I routine into a PL/I program, the main item of consideration in the area of storage management will be the DSA. If the routine is to be used in a multitasking environment, the code should be reentrant and should look like any other PL/I routine. You should ensure that PL/I conventions are followed, in particular setting the proper flags in the DSA and providing a pseudo-register for the routine. The DSA of the routine must be stored in the pseudo-register by the routine. The routine name (the CSECT name) should be seven characters long, in accordance with the naming convention for pseudo-registers. You can request the DSA by the following code:

```
L     15,=V(IHESADA)
LA    0,100
BALR  14,15
MVI   0(13),X'80'
```

The second load instruction loads the length of the DSA (note that 72 bytes is not enough; the minimum length of the DSA is 100 bytes).

On return from IHESADA, general register 13 will contain the address of the DSA. The reason for using IHESADA is that the DSA is now chained to the other DSA's and the PRV in the standard PL/I way. The flag byte will be used by IHESAFA in returning control. The coding for returning control is shown below:

```
L     15,=V(IHESAFA)
BR    15
```

When the above instructions are executed, the DSA is freed and control is returned to the caller.

To request a pseudo-register for the routine, append a 'B' to the CSECT name as an eighth character and use the result as the label of a DXD instruction. For example:

```
SUBRUTN  CSECT
SUBRUTNB DXD A
```

Code the SAVE register macro-instruction thus:

```
SAVE (14,12),,*
```

The asterisk causes the CSECT name to be coded into the program for easy identification in a main storage dump or snapshot. For intermodule trace routines, you should use the chain-back and chain-forward of the DSAs.

Arguments and Parameters

The (F) compiler generally conforms to the IBM System/360 Operating System standard calling sequence, as follows:

1. Arguments are passed by name, not by value.

2. General register 1 contains the address of the first address constant

in the parameter list, except when the main procedure is called.

3. The caller provides a register save area addressed via general register 13. The invoked program stores the contents of the general registers of the invoking program in this area.

4. The invoked program loads into general register 13 the address of a save area that can be used by any program that it invokes.

5. On return to the invoking program, general registers 2 through 13, the program mask, and the PICA will appear unchanged. General register 14 will point to the return address in the invoking program (set by a BALR instruction). General registers 0, 1, and 15, the floating-point registers, and the condition code may be changed.

The main difference between PL/I argument passing and that of other languages is that the addresses in the list addressed by general register 1 do not necessarily point to the data items that each represents. The list passed by the PL/I module is the same as that of other languages only for arithmetic element data items. In all other cases, the address in the list points to a dope vector. (The various dope vectors are discussed briefly above.)

In the following example, these abbreviations are used:

SDV	String Dope Vector
ADV	Array Dope Vector
STDV	Structure Dope Vector
SADV	String Array Dope Vector

Example:

```
DCL A FIXED DECIMAL,
    B FIXED BINARY,
    C FLOAT DECIMAL,
    D FLOAT BINARY,
    E PICTURE '999V99',
    F PICTURE 'XXXAAA',
    G CHARACTER(10),
    H BIT(16);
DCL 1 I(10),
      2 J FIXED DECIMAL,
      2 K CHAR(12);
DCL 1 L,
      2 M CHAR(4),
      2 N CHAR(5);
DCL P(5) FIXED,
    Q(5) CHAR(2),
    R(5,4) FIXED;
CALL SUB(A,B,C,D,E,F,G,H,I,I.J,I.J(1),L,
         L.M,P,P(1),Q,Q(1),R,R(1,1));
```

The first DECLARE statement above refers to element variables. A through E are arithmetic; the rest are strings. The second DECLARE statement defines an array of structures; the third a structure, and the fourth a number of arrays. The CALL statement generates the following parameter list; general register 1 will contain the address of the first address constant in the list;

```
PARM DC A(A)
     DC A(B)
     DC A(C)
     DC A(D)
     DC A(E)
     DC A(SDV of F)
     DC A(SDV of G)
     DC A(SDV of H)
     DC A(STDV of I)
     DC A(ADV of I.J)
     DC A(I.J(1))
     DC A(STDV of L)
     DC A(SDV of L.M)
     DC A(ADV of P)
     DC A(P(1))
     DC A(ADV of Q)
     DC A(SDV of Q(1))
     DC A(ADV of R)
     DC A(R(1,1))
```

There are several points to note about this parameter list. For variables A, B, C, D, and E, the parameter list is identical to those of object programs for other languages compiled under the IBM System/360 Operating System; this is because each is an arithmetic element variable. Similarly, even though they belong to arrays, I.J(1), P(1), and R(1,1) are passed by their actual addresses; their subscripts are included in the argument list of the CALL statement and they each identify a single element of an arithmetic array, so they are treated as elements. (This would be so even if the subscripts were expressions instead of constants; at execution time, a specific element would be referred to.) None of the other arguments represents an arithmetic element, and so they are represented by their respective dope vectors.

Note particularly the array of structures, I. The dope vector for I consists of two dope vectors: the ADV for I.J and the SADV for I.K. I itself, therefore, is not really treated as an array, but more as a structure with its dimensionality carried down to the elements of the structure. In this case, the multipliers used in the dope vectors will be the same for both I.J and I.K, and will be the length of a single J (3 bytes) plus the length of a single K (12 bytes), i.e., 15. The virtual origin of K is 3 bytes from the virtual origin of J. Had the item I been declared:

```
DCL 1 I,
      2 J(10) FIXED DECIMAL,
      2 K(10) CHAR(12);
```

then I would be a structure of arrays rather than an array of structures. The format of the dope vector for I would be the same, but the multipliers and virtual origins would be different. The multiplier for J would be 3, that for K would be 12, and the virtual origin of K would be 30 bytes from that of J.

If you are writing assembler language modules that will be called from PL/I programs, note the following points:

1. When a dope vector is passed for a string element, only another load register instruction is required to get the address of the actual data.

2. As shown in the examples of I.J(1), P(1), and R(1,1) above, passing the first element of an array or structure rather than the array or structure itself will avoid calculations and additional manipulation that would otherwise be necessary to obtain the actual data. This is possible only if the data characteristics are known in the called program.

3. An ADV contains the virtual origin of the array, not the actual origin; also, arrays are stored in row major order, not column major order.

COMMUNICATION WITH OTHER LANGUAGES

This section deals with the problems of calling non-PL/I modules from PL/I programs and vice versa. The main topics are passing data items between modules written in different languages and establishing the appropriate environment; also discussed are the use of function references between languages, and the invocation of user-defined PL/I ON-units from assembler language subroutines and functions.

Passing Data Items

When data items are to be passed from PL/I modules to non-PL/I modules and vice versa, the main considerations are:

1. The use of PL/I dope vectors

2. Data format differences

3. Data mapping differences

Only the first item is discussed in this section, since the other two have been discussed earlier in the chapter, under the heading 'Data Set Interchange.'

One way to solve the problem of dope vectors is to overlay arithmetic based variables in the PL/I program onto the beginning of items that require dope vectors (i.e., all items except arithmetic elements); thus the parameter can be made to look like an arithmetic element in the parameter list. Note that a based variable used in this way should not normally be referred to anywhere except in a call to a non-PL/I module. For example, if a character string is to be passed from a PL/I module:

```
DCL STRINGA CHAR(20),
    PARMA FIXED DEC(1,0) BASED(P);
P=ADDR(STRINGA);
CALL SUB(PARMA);
```

In this case, the parameter list will consist of an address constant that directly addresses PARMA, an arithmetic element variable. But since the pointer P points to the first character position of STRINGA, the routine SUB will have access to STRINGA without concern for the STRINGA dope vector.

Similarly, if the string is to be passed to a PL/I module from a non-PL/I module, the PL/I coding could be as follows:

```
PL1SUB: PROC(PARMA);
        DCL PARMA FIXED DEC(1,0),
            STRINGA CHAR(20) BASED(P);
        P=ADDR(PARMA);
```

Now the dope vector for STRINGA will point to PARMA. Since PARMA is an arithmetic element, the PL/I program will expect the parameter list to contain the address of PARMA.

Another way of communicating data items between FORTRAN and PL/I modules is to use named common storage for the item. You can do this by declaring the identifier STATIC EXTERNAL in the PL/I module and COMMON in the FORTRAN module. This results in an external control section. For example:

FORTRAN

```
COMMON/COMAR/DVAR,I,A,CHR,IRAY
DIMENSION DVAR(7),CHR(3),IRAY(100)
```

PL/I

```
DCL 1 COMAR STATIC EXTERNAL,
      2 I FIXED BIN(31,0),
      2 A FLOAT DEC,
      2 CHR CHAR(12),
      2 IRAY(100) FIXED BIN(31,0);
```

The actual storage layout would be as shown in Figure 15-4.

Note the following points in connection with this method of communication:

1. There may be padding between the PL/I dope vector and the start of the structure. The dope vector is always at the start of the control section and is therefore always located on a doubleword boundary; it will consist of a number of fullwords. Therefore there could be up to seven unused bytes between the dope vector and the structure if the member having the highest alignment requirement is not the first member of the structure. In the above example, there is no padding, since the structure is word-aligned and starts with a fullword. If I had been halfword binary, there would be two bytes of padding between the dope vector and the structure. This could be accommodated by a dummy variable of length two in the FORTRAN COMMON statement immediately after DVAR; alternatively, DVAR could be declared as INTEGER*2 and its dimension increased to 15.

2. When there are two or more identical external references, the linkage editor acts on the first one encountered. Since the PL/I dope vector must be completed, the PL/I module must precede the FORTRAN module on input to the linkage editor. (It need not necessarily be executed first; you can use the ENTRY statement at the end of the JCL for the link-edit to ensure that the correct module gets initial control.)

Establishing the Environment

If one procedure only is to be called from a non-PL/I routine, and if releasing the PL/I environment on exit from the procedure does not adversely affect the execution of the program, the PL/I procedure should be given the MAIN option. The PL/I procedure will be entered by the statement CALL IHESAPD. (Since the name IHESAPD has more than six characters, FORTRAN programs must have a dummy name for IHESAPD that is six or less characters long. You can change this name to IHESAPD at link-edit time using the linkage editor CHANGE statement immediately before the FORTRAN module or its INCLUDE statement.)

If more than one PL/I procedure is to be called from non-PL/I routines or if FORTRAN or COBOL subroutines or FORTRAN functions are to be invoked from PL/I, an assembler language interface routine must be used. The main purpose of this interface routine is to save the environment of the language that called it and to establish the environment of the language to which it is to give control.

If you are calling multiple PL/I procedures from COBOL or FORTRAN, your interface routine should perform certain functions:

1. Assuming that calls to PL/I modules from other languages will not be nested, a 'first-time-through' routine can be executed to call the library module IHESAP, which establishes the PL/I environment, at entry-point IHESAPD. This would require that the interface routine contain a control section called IHEMAIN that contains the address to which IHESAP is to transfer control after initializing the environment.

2. The pointer to the PRV, returned by IHESAP in general register 12, should be saved for future executions of the interface routine. Note that the address returned in register 1 by the SPIE macro instruction issued by IHESAP has been saved by the PL/I library. If it is required, in order to reissue the SPIE macro instruction of the calling program, it may be obtained by the following code:

```
L   Rx,0(12)
ORG *-2
DC  QL2(IHEQLSA)
L   Rx,12(Rx)
```

3. For all executions after the first, the parameter list pointer in general register 1 should be saved and the SPIE macro instruction for PL/I should be issued, specifying IHEERRA as the routine for handling interrupts. The address returned by the SPIE macro instruction in register 1 should be saved as it will be required to reset the COBOL or FORTRAN SPIE macro instruction on return from the PL/I procedure.

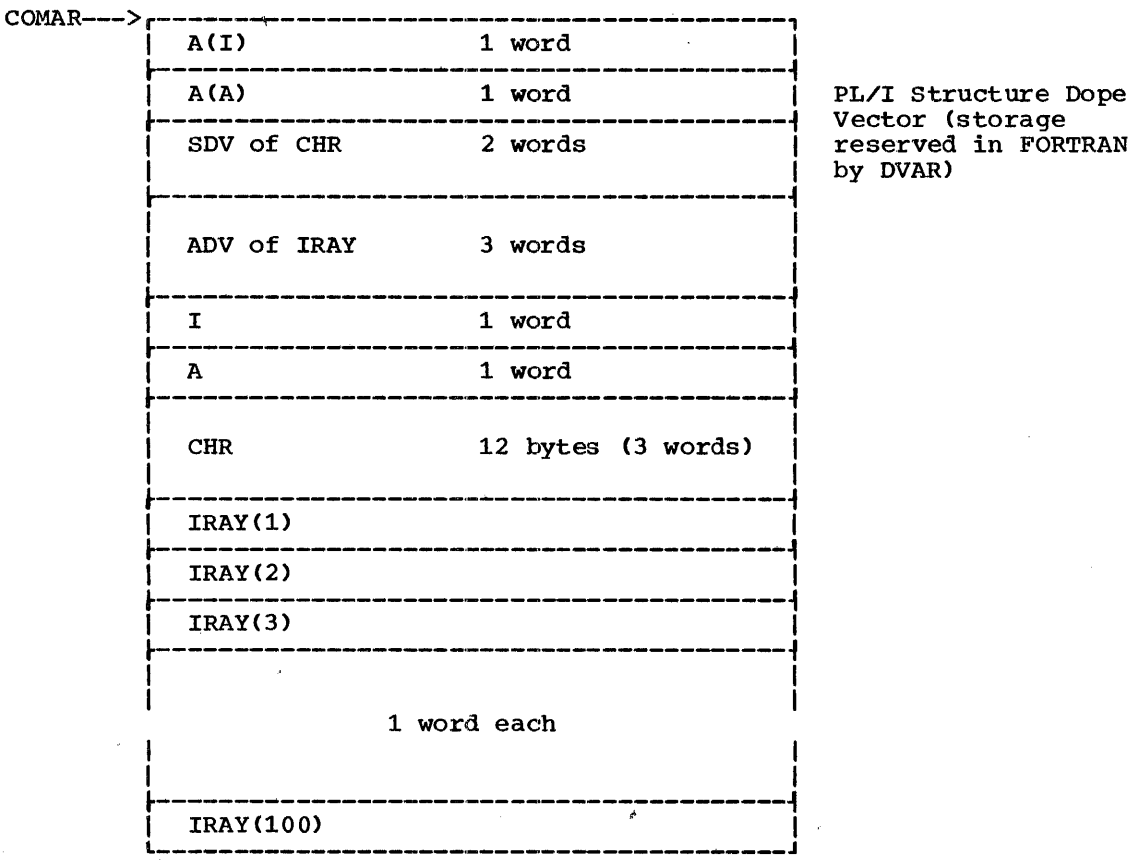


Figure 15-4. PL/I-FORTRAN: Example of Named Common Storage

4. Provided that general register 12 points to the PRV, the PL/I entry point can be called. If the argument list from COBOL or FORTRAN was of the form:

   (PL/I-entry-point,parm-1,parm-2,...)

   then general register 1 (whose contents were saved in step 3) will point to the PL/I entry address constant. If the value 4 is now added to general register 1, the result will be the address of the parameter list that the PL/I module expects.

5. On return from the PL/I module, the COBOL or FORTRAN SPIE is reset and control is passed back to the original calling module.

If a FORTRAN or COBOL module is to be called from a PL/I program, basically the same procedure as outlined above can be used as an interface, the difference being:

1. In calling a COBOL module no initialization of an environment is required except that the SPIE macro instruction must be re-issued.

2. In calling a FORTRAN module, initialization of the environment is carried out by invoking the FORTRAN library as follows:

   ```
   L 15,=V(IBCOM#)

   BAL 14,64(15)
   ```

Communication between assembler language and PL/I modules is subject to the same interface problems, and the above concepts apply. However, additional conventions should be observed by an assembler routine that invokes a PL/I routine and is itself invoked by another PL/I routine. The conventions are given in the preceding section "Storage Organization and Control Blocks" under the sub-heading "Diagnostic Storage Area (DSA)".

Function References

The basic difference between a subroutine and a function is that a subroutine is invoked by a CALL statement and does not actually return a value (although it may

change the value of a variable known to the calling program), whereas a function is invoked by a function reference which can be used in the same way as an expression; the function always returns a value, and this value is represented by the function reference. COBOL does not have function references, and so the following discussion does not apply to COBOL-PL/I communication.

A function written in another language can be declared with the ENTRY and RETURNS attributes in a PL/I program. Then, whenever the function name occurs in the program, control will be passed to the user-written function of that name. On entry to the function, normal linkage should be performed.

At execution time, the difference between a PL/I function reference and a subroutine call is that the function parameter list will have an extra item representing a temporary into which the function is to place its result. The address, depending on the data characteristics declared in the RETURNS attribute in the invoking program, will either be direct or will be that of a dope vector (according to the same rules as for normal parameters). Just as with entry declarations for PL/I-written procedures, you could use the GENERIC attribute to specify alternative entry points to an assembler language function, based on the declared attributes of the arguments. The assembler language function need only specify each name as an entry point.

The method employed to return the value from a function to the invoking procedure differs between FORTRAN and PL/I. A FORTRAN function returns its value in general register 0 (if integer or logical), floating-point register 0 (if real), or floating-point registers 0 and 1 (if complex). In PL/I, as already mentioned, an additional argument is automatically provided for the returned value. To resolve this difference in function communication, an assembler language interface routine must be used. When a FORTRAN program invokes a PL/I function, this routine would create a new argument list consisting of the original list and an additional argument. The routine would pick up the value returned in this additional argument when the function is complete, and place the value in the appropriate register(s) before passing control back to the FORTRAN program. When a PL/I program invokes a FORTRAN function, the assembler language routine would obtain the result from the register and place it in the additional argument. Separate entry points to the assembler language interface routine would be necessary for different data types since the location of the result depends on its characteristics.

User-Defined Conditions in Assembler Language Subroutines and Functions

When you write a PL/I program, you can provide an ON-unit for any circumstance you wish to test for. The ON-unit is defined by any ON CONDITION(condition-name) statement, the condition name being one of your own choosing. (Note that since a user-defined condition is given the EXTERNAL attribute, the name must be no more than seven characters long.) The ON-unit can be invoked by a SIGNAL statement naming the condition.

If a PL/I program invokes an assembler language module, and such a user-defined condition occurs during the execution of the module, the equivalent of the appropriate SIGNAL statement will be needed in the module in order to invoke the ON-unit. To provide this, the assembler language routine must generate a parameter list of one word; byte one of the word contains X'50' and the second through fourth bytes are an identifying field containing the address of an external control section that has the same name as the user-defined condition. General register 1 is then set to point to this parameter list, and the routine calls the PL/I library entry IHEERRD. After execution of the ON-unit, control will be returned to the instruction following the IHEERRD call. An example of a 'SIGNAL' to a user-defined condition named COND would be as follows:

```
        L    15,=V(COND)
        ST   15,PARM
        MVI  PARM,X'50'
        LA   1,PARM
        L    15,=V(IHEERRD)
        BALR 14,15
        .
        .
        .
PARM    DS   F
```

The register save area for the routine should look like a PL/I DSA. That is, the first byte of the first word should contain X'80', and bytes two through four should contain the save area length. The second and third words should be used for the chain-back and chain-forward addresses respectively.

PART 3: Appendixes

Appendix A: Programming Examples

Each of the following examples reproduces the complete printed listing. Each listing is preceded by a description of the example and the listing, including references to parts of the manual that deal with the features being illustrated. All the source statements, job control statements, and data cards are listed as they would appear in the job stream.

For easy reference to the listing, the description uses the circled numbers that have been added to the top right-hand corner of each page of the listing.

Example 1: Simple PL/I Program

This example illustrates the use of the cataloged procedure PL1LFCLG to compile, link-edit, and execute a simple PL/I program that uses only punched-card input and printed output. The job control statements it includes are described in Chapter 2.

```
//J013PGEA JOB
//  EXEC PL1LFCLG
//PL1L.SYSIN DD *
 EX001:  PROCEDURE OPTIONS(MAIN);
         DECLARE(A,B,C) FIXED DECIMAL(3);
 NEXT:   GET FILE(SYSIN) DATA(A,B);
         IF A ¬= 999 THEN DO;
           C=A+B;
           PUT FILE(SYSPRINT) SKIP
                    DATA(A,B,C);
           GO TO NEXT;
           END;
         END EX001;
/*
//GO.SYSIN DD *
A=131 B=75;
A=2 B=907;
A=-14 B=14;
A=341 B=429;
A=-245 B=102;
A=999 B=100;
/*
```

LISTING

Page 1: The JOB statement, and a list of the addresses of the units to which the data sets for the first job step (PL1L) were assigned.

Pages 2-4: The listing produced by the PL/I (F) compiler, which is described fully in Chapter 5. The PL/I source statements appear on page 3 of the full listing (page 2 of the compiler listing).

Page 5: The disposition of the data sets used by the compiler in the first job step, and a list of the addresses of the units to which the data sets for the second job step (LKED) were assigned.

Pages 6-8: The listing produced by the linkage editor; this is described fully in Chapter 6.

Page 9: The disposition of the data sets used by the linkage editor in the second job step, and a list of the addresses of the units to which the data sets for the third job step (GO) were assigned.

Page 10: The printed output produced by the example program.

Page 11: The disposition of the data sets used by the example program in the third job step.

```
//J013PGEA JOB
   JOB LOGGED ON AT IBM'S HURSLEY LABS      OS/360 MVT 18          CLOCK=13.38.18   DATE=69.273   MACHINE=J1
IEF236I ALLOC. FOR J013PGEA PL1L
IEF237I SYSPRINT ON 293
IEF237I SYSLIN   ON 342
IEF237I SYSUT3   ON 343
IEF237I SYSUT1   ON 344
IEF237I SYSIN    ON 290
```

(1)

```
VERSION 5.0                                    OS/360 PL/I COMPILER (F)                    PAGE     1
                                                                                           DATE 69.273

PL/I F COMPILER OPTIONS SPECIFIED ARE AS FOLLOWS--

     LOAD,NODECK,SIZE=100K

THE COMPLETE LIST OF OPTIONS USED DURING THIS COMPILATION IS--   EBCDIC
                                                                 CHAR60
                                                               NOMACRO
                                                                 SOURCE2
                                                               NOMACDCK
                                                                 COMP
                                                                 SOURCE
                                                               NOATR
                                                               NOXREF
                                                               NOEXTREF
                                                               NOLIST
                                                                 LOAD
                                                               NODECK
                                                                 FLAGW
                                                                 STMT
                                                                 SIZE=0102400
                                                                 LINECNT=055
                                                                 OPT=02
                                                                 SORMGIN=(002,072,001)
                                                               NOEXTDIC
                                                                 NEST
                                                                 OPLIST

*OPTIONS IN EFFECT*     EBCDIC,CHAR60,NOMACRO,SOURCE2,NOMACDCK,COMP,SOURCE,NOATR,NOXREF,NOEXTREF,NOLIST,LOAD,
*OPTIONS IN EFFECT*     NODECK,FLAGW,STMT,SIZE=0102400,LINECNT=055,OPT=02,SORMGIN=(002,072,001),NOEXTDIC,
*OPTIONS IN EFFECT*     NEST,OPLIST
```

(2)

```
       EX001:  PROCEDURE OPTIONS(MAIN);                                                    PAGE     2

STMT LEVEL NEST
  1               EX001:  PROCEDURE OPTIONS(MAIN);
  2     1                 DECLARE (A,B,C) FIXED DECIMAL(3);
  3     1         NEXT:   GET FILE(SYSIN) DATA(A,B);
  4     1                 IF A¬=999 THEN DO;
  6     1     1             C=A+B;
  7     1     1             PUT FILE(SYSPRINT) SKIP DATA(A,B,C);
  8     1     1             GO TO NEXT;
  9     1     1             END;
 10     1                 END EX001;
```

(3)

```
          EX001:  PROCEDURE OPTIONS(MAIN);                                                    PAGE      3

          STORAGE REQUIREMENTS.
          ---------------------

          THE STORAGE AREA FOR THE PROCEDURE LABELLED EX001 IS 192 BYTES LONG.

          THE PROGRAM CSECT IS NAMED EX001    AND IS 200 BYTES LONG.

          THE STATIC  CSECT IS NAMED **EX001A AND IS 219 BYTES LONG.

*STATISTICS*      SOURCE RECORDS =         9,PROG TEXT STMNTS =        10,OBJECT BYTES =       200

 NO ERRORS OR WARNINGS DETECTED.

AUXILIARY STORAGE WILL NOT BE USED FOR DICTIONARY WHEN SIZE =  65K

COMPILE TIME          .02 MINS

ELAPSED TIME          .38 MINS
```

4

```
IEF285I   SYS69273.T132652.SV0C0.J013PGEA.R0000027      SYSOUT
IEF285I   VOL SER NOS= M65293.
IEF285I   SYS69273.T132652.RV0C0.J013PGEA.LOADSET       PASSED
IEF285I   VOL SER NOS= 231422.
IEF285I   SYS69273.T132652.RV000.J013PGEA.R0000028      DELETED
IEF285I   VOL SER NOS= 231423.
IEF285I   SYS69273.T132652.RV000.J013PGEA.R0000029      DELETED
IEF285I   VOL SER NOS= 231424.
IEF285I   SYS69273.T132652.RV000.J013PGEA.S0000030      SYSIN
IEF285I   VOL SER NOS= M65295.
IEF285I   SYS69273.T132652.RV000.J013PGEA.S0000030      DELETED
IEF285I   VOL SER NOS= M65295.
   END OF STEP 'PL1L    '   JOB 'J013PGEA'    STEPTIME=00.00.02  CLOCK=13.38.55   DATE=69.273   RETURN CODE=0000
IEF236I ALLOC. FOR J013PGEA LKED
IEF237I SYSLIB   ON 295
IEF237I          ON 295
IEF237I          ON 1C0
IEF237I SYSLMOD  ON 342
IEF237I SYSUT1   ON 343
IEF237I SYSPRINT ON 294
IEF237I SYSLIN   ON 342
```

5

```
 F88-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED MAP,LIST
          VARIABLE OPTIONS USED - SIZE=(104448,59392)              DEFAULT OPTION(S) USED
IEW0461  IHEDDJA
IEW0461  IHEDDPA
IEW0461  IHEDDPB
IEW0461  IHEDDPC
IEW0461  IHEDDPD
IEW0461  IHEUPBA
IEW0461  IHEUPBB
IEW0461  IHEVPAA
IEW0461  IHEVPCA
IEW0461  IHEVFBA
IEW0461  IHEVFDA
IEW0461  IHEVFEA
IEW0461  IHEVKBA
IEW0461  IHEVKCA
IEW0461  IHEVKFA
IEW0461  IHEVKGA
IEW0461  IHEVSBA
IEW0461  IHEVSFA
IEW0461  IHEVSEB
IEW0461  IHEM91A
IEW0461  IHEM91B
IEW0461  IHEM91C
IEW0461  IHETERA
IEW0461  IHEVCSA
IEW0461  IHEKCDB
IEW0461  IHEDNBA


                                    MODULE MAP

  CONTROL SECTION                          ENTRY

    NAME     ORIGIN   LENGTH      NAME   LOCATION     NAME   LOCATION     NAME   LOCATION     NAME   LOCATION

  EX001         00       C8
  **EX001A      C8       DB
  IHEMAIN      1A8        4
  IHENTRY      1B0        C
  IHESPRT      1C0       38
  SYSIN        1F8       38
  IHEDDI   *   230      53C
                                  IHEDDIA    230    IHEDDIB    232
  IHEDDO   *   770      288
                                  IHEDDOA    770    IHEDDOB    772    IHEDDOC    774    IHEDDOD    776
                                  IHEDDOE    778
  IHEIOA   *   9F8      16A
                                  IHEIOAA    9F8    IHEIOAB    9FA    IHEIOAC    9FC    IHEICAD    9FE
                                  IHEIOAT    AE8
  IHEIOB   *   B68      1E4
                                  IHEIOBA    B68    IHEIOBB    B70    IHEIOBC    B78    IHEICBD    B80
                                  IHEIOBE    B88    IHEIOBT    C74
  IHESAP   *   D50      AB8
```

6

NAME		ORIGIN	LENGTH
IHEDCN	*	1808	1EF
IHEDMA	*	19F8	F8
IHEDNC	*	1AF0	284
IHEUPA	*	1D78	E8
IHEVFA	*	1E60	16C
IHEVPB	*	1FD0	1A2
IHEVPD	*	2178	105
IHEVPE	*	2280	26C
IHEVPF	*	24F0	50
IHEVSC	*	2540	AC
IHEERR	*	25F0	729
IHEIOF	*	2D20	2DC
IHELDI	*	3000	858
IHEVFC	*	3858	26
IHEVPG	*	3880	229
IHEVPH	*	3AB0	B4
IHEVQB	*	3B68	494
IHEVQC	*	4000	268
IHEBEG	*	4268	80
IHEIOP	*	42E8	1EB
IHELDO	*	44D8	418
IHEOCL	*	48F0	554
IHEPRT	*	4E48	2C8

NAME	LOCATION	NAME	LOCATION	NAME	LOCATION	NAME	LOCATION
IHESADA	D50	IHESAPC	D6A	IHESAPD	D72	IHESAPA	D7A
IHESAPB	D82	IHESADF	D8A	IHESADB	D92	IHESADE	D9A
IHESAFC	DA2	IHESAFA	DAA	IHESAFB	DB2	IHESAFD	DBA
IHESARA	DC2	IHESAFQ	DCA	IHESARC	1586	IHESADD	1674
IHESAFF	16AE						
IHEDCNA	1808	IHEDCNB	180A				
IHEDMAA	19F8						
IHEDNCA	1AF0						
IHEUPAA	1D78	IHEUPAB	1DE2				
IHEVFAA	1E60						
IHEVPBA	1FD0						
IHEVPDA	2178						
IHEVPEA	2280						
IHEVPFA	24F0						
IHEVSCA	2540						
IHEERRD	25F0	IHEERRC	25FA	IHEERRB	2604	IHEERRA	260E
IHEERRE	2C86						
IHEIOFB	2D20	IHEIOFA	2D22	IHEITAZ	2FBE	IHEITAX	2FCA
IHEITAA	2FDE						
IHELDIA	3000	IHELDIB	3002	IHELDIC	3004	IHELDID	3006
IHEVFCA	3858						
IHEVPGA	3880						
IHEVPHA	3AB0						
IHEVQBA	3B68						
IHEVQCA	4000						
IHEBEGN	4268	IHEBEGA	42A8				
IHEIOPA	42E8	IHEIOPB	42EA	IHEIOPC	42EE		
IHELDOA	44D8	IHELDOB	44DA	IHELDOC	44DE		
IHEOCLA	48F0	IHEOCLB	48F2	IHEOCLC	48F4	IHEOCLD	48F6

(7)

```
   NAME     ORIGIN  LENGTH                NAME    LOCATION     NAME    LOCATION     NAME    LOCATION     NAME    LOCATION
                                        IHEPRTA     4E48     IHEPRTB     4E4A
  IHESIZ  *   5110       C             IHESIZE     5110
  IHETAB  *   5120       C             IHETABS     5120
  IHEVCA  *   5130      10A            IHEVCAA     5130
  IHEVQA  *   5240      E6             IHEVQAA     5240
  IHEVSD  *   5328     1A0             IHEVSDA     5328     IHEVSDB     532A

 PSEUDO REGISTERS

   NAME     ORIGIN  LENGTH       NAME     ORIGIN  LENGTH       NAME     ORIGIN  LENGTH       NAME     ORIGIN  LENGTH
  IHEQINV       00       4    IHEQERR          4       4    IHEQTIC          8       4    **EX001B         C       4
  IHEQSPR       10       4    SYSIN           14       4    IHEQLSA         18       4    IHEQLW0         1C       4
  IHEQLW1       20       4    IHEQLW2         24       4    IHEQLW3         28       4    IHEQLW4         2C       4
  IHEQLWE       30       4    IHEQLCA         34       4    IHEQVDA         38       4    IHEQFVD         3C       4
  IHEQFOP       40       4    IHEQCFL         44       8    IHEQADC         4C       4    IHEQLPR         50       4
  IHEQSLA       54       4    IHEQSAR         58       4    IHEQLWF         5C       4    IHEQRTC         60       4
  IHEQSFC       64       4    IHEQXLV         68       8    IHEQEVT         70       8

 TOTAL LENGTH OF PSEUDO REGISTERS      78
 ENTRY ADDRESS      180
 TOTAL LENGTH      54C8

****GO        DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET

                                        DIAGNOSTIC MESSAGE DIRECTORY

  IEW0461 WARNING - SYMBOL PRINTED IS AN UNRESOLVED EXTERNAL REFERENCE, NCAL WAS SPECIFIED.
```

(8)

```
IEF285I   SYS1.PL1LIB                                    KEPT
IEF285I   VOL SER NOS= 2301C0.
IEF285I   SYS69273.T132652.RV0C0.J013PGEA.GOSET          PASSED
IEF285I   VOL SER NOS= 231422.
IEF285I   SYS69273.T132652.RV000.J013PGEA.R0000031       DELETED
IEF285I   VOL SER NOS= 231423.
IEF285I   SYS69273.T132652.SVC00.J013PGEA.R0000032       SYSOUT
IEF285I   VOL SER NOS= M65294.
IEF285I   SYS69273.T132652.RV000.J013PGEA.LOADSET        DELETED
IEF285I   VOL SER NOS= 231422.
   END OF STEP 'LKED    '   JOB 'J013PGEA'    STEPTIME=00.00.02  CLOCK=13.39.33   DATE=69.273   RETURN CODE=0004
IEF236I ALLOC. FOR J013PGEA GO
IEF237I PGM=*.DD ON 342
IEF237I SYSPRINT ON 294
IEF237I SYSIN    ON 290
```

(9)

```
A=   131              B=    75              C=   206;
A=     2              B=   907              C=   909;
A=   -14              B=    14              C=     0;
A=   341              B=   429              C=   770;
A=  -245              B=   102              C=  -143;
```

(10)

```
IEF285I   SYS69273.T132652.RV000.J013PGEA.GOSET          PASSED
IEF285I   VOL SER NOS= 231422.
IEF285I   SYS69273.T132652.SV000.J013PGEA.R0000033       SYSOUT
IEF285I   VOL SER NOS= M65294.
IEF285I   SYS69273.T132652.RV000.J013PGEA.S0000034       SYSIN
IEF285I   VOL SER NOS= M65295.
IEF285I   SYS69273.T132652.RV000.J013PGEA.S0000034       DELETED
IEF285I   VOL SER NOS= M65295.
   END OF STEP 'GO      '   JOB 'J013PGEA'     STEPTIME=00.00.01  CLOCK=13.40.47   DATE=69.273   RETURN CODE=0000
IEF285I   SYS69273.T132652.RV000.J013PGEA.GOSET          DELETED
IEF285I   VOL SER NOS= 231422.
   END OF JOB  'J013PGEA'  I/O TIME=00.00.08   CPUTIME=00.00.06  CLOCK=13.40.48   DATE=69.273   SYSTEM PACKS 300 AND 311
```

(11)

Example 2: Compiler and Linkage Editor Listings

```
//J063PGEX JOB
//COLEEX EXEC PL1LFCLG,PARM.PL1L='L,E,A,X,M,S2,NT',
//             PARM.LKED='LIST,XREF,OVLY',
//             COND.GO=((9,LT,LKED),(15,LT,PL1L))
//PL1L.SYSIN DD *
 P1:      PROC OPTIONS(MAIN);
          %DCL AA CHAR,(V1,V2,V3,V4) FIXED;
          %AA='IF ANS>COMP THEN WORDS=''GREATER THAN'';
              ELSE IF ANS<COMP THEN WORDS=''LESS THAN'';
                   ELSE WORDS=''EQUAL TO'';
              PUT SKIP DATA(ANS,COMP);
              PUT SKIP EDIT(''ANSWER'',WORDS,''COMPARATOR'')(A);';
          %V1=1; %V3=2; %V4=4;
          DCL (ARG(2,2) STATIC INIT(V1,V2,V3,V4),(COMP,ANS)EXT)
            FIXED DEC(2,1),
              WORDS CHAR(12) VAR,
                (P2,P3) ENTRY EXT;
          DO I=1,2;
            DO J=1,2;
              COMP=ARG(I,J)/2;
              CALL P2;
              AA
              CALL P3;
              AA
          END P1;
*PROCESS('E,NT');
 P2:      PROC;
          DCL (COMP,ANS) FIXED DEC(2,1) EXT;
          ON ZDIV BEGIN;
            ANS=0;
            PUT SKIP LIST('NEXT COMMENT INVALID');
            GO TO OUT;
            END;
          ANS=1/COMP;
 OUT:     END P2;
*PROCESS('E');
 P3:      PROC;
          DCL (COMP,ANS) FIXED DEC(2,1) EXT;
          ANS=COMP*COMP;
          END P3;
/*
//LKED.SYSIN DD *
 OVERLAY X
 INSERT P2
 INSERT IHELDO
 OVERLAY X
 INSERT P3
/*
```

This example illustrates all the components of the listings produced by the compiler and the linkage editor. The listings themselves are described in Chapters 5 and 6.

The program used for this example is not intended to be a realistic illustration of an application of PL/I, but is designed merely to employ all those compiler and linkage-editor facilities that contribute to the listings. The program comprises three external procedures (P1, P2, and P3), which are compiled in a single execution of the compiler (using the batched-processing facility), link-edited into an overlay load module, and then executed.

The input statements include only one EXEC statement, which invokes the cataloged procedure PL1LFCLG. This statement changes the COND parameter in the EXEC statement for the procedure step GO to permit execution of the load module despite the severe error that is detected by the compiler preprocessor.

LISTING

Page 1:
1. The JOB and EXEC statements, exactly as they appear in the input stream.
2. The job control statements of the cataloged procedure step PL1L (compilation). Each of these statements is prefixed XX.
3. The DD statement PL1L.SYSIN, which is added to the job control statements for the cataloged procedure step PL1L. The data that follows this DD statement in the input stream is not listed by the job scheduler.
4. A list of the addresses of the units to which the compiler data sets are assigned.

Page 2: Start of the compiler listing: list of options applicable to the compilation of procedure P1. Note that the SIZE option specifies the actual amount of storage allocated to the compiler.

Page 3: Source statements for procedure P1, exactly as they appear in the input stream. These statements form the input data for the compiler preprocessor.

Page 4: Error message from the preprocessor. The zero value assigned to V2 does not prevent execution of the program.

Page 5: Output from the preprocessor (modified source statements for procedure P1).

Page 6: Attribute and cross-reference table for procedure P1.

Page 7: Aggregate-length table for procedure P1.

Page 8: Storage requirements for procedure P1. Note that these quantities do not include the storage that will be occupied by the subroutine library modules that will be included by the linkage editor or loaded dynamically during execution of the object program.

Page 9: External symbol dictionary for procedure P1.

Page 10: Static internal storage map for procedure P1.

Page 11-13: Object program listing for procedure P1.

Page 14:
1. Warning message from the compiler.
2. Statement of the time taken for compilation of procedure P1, including the preprocessor stage, and the elapsed time since the start of compilation. Note that, since the program was executed under the MVT option of the operating system, the compile time is only a fraction of the elapsed time.

Page 15: List of options applicable to the compilation of procedure P2.

Page 16: Source statements for procedure P2, exactly as they appear in the input stream.

Page 17: Storage requirements for procedure P2.

Page 18: External symbol dictionary for procedure P2.

Page 19:
1. Warning messages from the compiler. The omission of the option MAIN from the PROCEDURE statement does not affect execution of the program because control will pass initially to procedure P1.
2. Statement of the time taken for the compilation of procedure P2 and the elapsed time since the start of the compilation of this procedure.

Page 20: List of options applicable to the compilation of procedure P3.

Page 21: Source statements for procedure P3, exactly as they appear in the input stream.

Page 22: Storage requirements for procedure P3.

Page 23: External symbol dictionary for procedure P3.

Page 24:
1. Warning message from the compiler.
2. Statement of the time taken for the compilation of

procedure P3 and the elapsed time since the start of the compilation of this procedure.

This is the last page of the compiler listing.

Page 25:

1. The dispositon of the data sets used by the compiler. Only the data set that contains the object module is passed to the next job step; it is identified by a system-generated name that ends with the temporary name LOADSET assigned to it in the DD statement SYSLIN.

2. The job scheduler END OF STEP message, including the return code supplied by the compiler. The code 0012 indicates that the compiler detected a severe error (the non-initialization of the preprocessor variable V2 in procedure P1).

3. The job control statements of the cataloged procedure step LKED (link-editing). Each of these statements is prefixed XX.

4. The DD statement LKED.SYSIN, which is added to the job control statements for the cataloged procedure step LKED. The data that follows this DD statement in the input stream (linkage-editor control statements) is not listed by the job scheduler.

5. A list of the addresses of the units to which the linkage-editor data sets are assigned.

Page 26: Start of the linkage-editor listing. This page contains the following:

1. Statements of the options specified, and of the default values supplied for the SIZE option.

2. The control statements processed by the linkage editor. These statements appear exactly as in the input stream; they specify the structure of the overlay module. The load module comprises three segments: the program control section for procedure P2 and library module IHELDO are in one overlay segment, and the program control section for P3 is in another; the rest of the load module is in the root segment. (Inspection of the external symbol dictionaries for procedures P1, P2, and P3 will reveal that, of the library modules to be link-edited into the load module, only IHELDO, which is called by P2, is required in an overlay segment and not in the root segment; accordingly, this is the only library module that can be moved out of the root segment.)

3. A list of the external references that were not to be resolved by the linkage editor. (See 'Diagnostic Message Directory' in Chapter 6.)

4. Start of the module map for the root segment. (The maps and cross-reference tables for all segments follow the heading 'Cross Reference Table'.) Note that the first control section in the root segment has the name $SEGTAB. This control section contains the segment table always created by the linkage editor for an overlay module; it includes control information for use by the operating system during execution of the module.

Page 27: Remainder of the module map, and the start of the cross-reference table, for the root segment. Note that the last control section in this segment has the name $ENTAB. This is another control section created by the linkage editor; it contains the entry table, which includes information used by the operating system to determine which segment is to be loaded next. The cross-reference table for the root segment continues through page 28 to the top of page 29.

Page 29:

1. Completion of the cross-reference table for the root segment.

2. Maps and cross-reference tables for the two overlay segments.

3. Start of the list of pseudo-registers.

Page 30:

1. Remainder of the list of pseudo-registers.

2. Statement of the disposition of the load module GO. (Refer to 'Control Statements and Errors' in Chapter 6.)

3. Diagnostic Message Directory. The only message refers to the list of modules external references that were not to be resolved. (See above.)

This is the end of the linkage-editor listing.

Page 31:

1. The disposition of the data sets used by the linkage editor. Only the data set that contains the load module is passed to the last job step; it is identified by a system generated name that ends with the temporary name GOSET assigned to it in the DD statement SYSLMOD. (The load module is the member GO of the temporary library GOSET.)

2. Job scheduler END OF STEP message, including the return code supplied by the linkage editor. The code 0004 indicates that the linkage editor has issued a warning message (referring to the references that were not resolved).

3. The job control statements of the cataloged procedure step GO (execution of the load module). Each of these statements is prefixed XX.

4. A list of the addresses of the units to which the data sets required during execution of the load module are assigned.

Page 32: The printed output from the PL/I program.

Page 33:

1. The disposition of the data sets used during execution of the load module. Note that the data set GOSET is not deleted until after the end of the job because step GO does not contain a DD statement that specifies its disposition.

2. The job scheduler END OF STEP message for the step GO.

3. Statement of the disposition of the data set GOSET.

4. Job scheduler END OF JOB statement.

```
//J063PGEX JOB (4056,N109),R.SMALL.MSGLEVEL=1,MSGCLASS=D,CLASS=D
//COLEEX EXEC PL1LFCLG,PARM.PL1L='L,E,A,X,M,S2,NT',                    /
//           PARM.LKED='LIST,XREF,OVLY',                               /
//           COND.GO=((9,LT,LKED),(1,5,LT,PL1L))
XXPL1L   EXEC PGM=IEMAA,PARM='LOAD,NODECK,SIZE=100K',REGION=108K        00020000

XXSYSPRINT DD  SYSOUT=A,DCB=(RECFM=VBA,LRECL=125,BLKSIZE=629)           00040000
XXSYSLIN   DD  DSNAME=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,            *00060000
XX             SPACE=(400,(50,50)),DCB=BLKSIZE=400                      00080000
XXSYSUT3   DD  UNIT=SYSDA,SPACE=(80,(250,250)),DCB=BLKSIZE=80           00100000
XXSYSUT1   DD  UNIT=SYSDA,SPACE=(1024,(60,60),,CONTIG),                 *00120000
XX             DCB=BLKSIZE=1024                                         00140000
//PL1L.SYSIN DD *
   JOB LOGGED ON AT IBM'S HURSLEY LABS     OS/360 MVT 18         CLOCK=11.55.04   DATE=69.273   MACHINE=J1
IEF236I ALLOC. FOR J063PGEX PL1L     COLEEX
IEF237I SYSPRINT ON 294
IEF237I SYSLIN   ON 342
IEF237I SYSUT3   ON 343
IEF237I SYSUT1   ON 344
IEF237I SYSIN    ON 290
```

(1)

Note: The cataloged procedure shown in this example is that in use at the Hursley Laboratories, and varies slightly from the standard cataloged procedure shown in Chapter 8.

```
VERSION 5.0                                  OS/360 PL/I COMPILER (F)                    PAGE     1
                                                                                         DATE 69.273

PL/I F COMPILER OPTIONS SPECIFIED ARE AS FOLLOWS--

     L,E,A,X,M,S2,NT

THE COMPLETE LIST OF OPTIONS USED DURING THIS COMPILATION IS--   EBCDIC
                                                                CHAR60
                                                                MACRO
                                                                SOURCE2
                                                              NOMACDCK
                                                                COMP
                                                                SOURCE
                                                                ATR
                                                                XREF
                                                                EXTREF
                                                                LIST
                                                                LOAD
                                                              NODECK
                                                                FLAGW
                                                                STMT
                                                                SIZE=0090112
                                                                LINECNT=055
                                                                OPT=02
                                                                SORMGIN=(002,072,001)
                                                              NOEXTDIC
                                                                NEST
                                                                OPLIST

*OPTIONS IN EFFECT*     EBCDIC,CHAR60,MACRO,SOURCE2,NOMACDCK,COMP,SOURCE,ATR,XREF,EXTREF,LIST,LOAD,
*OPTIONS IN EFFECT*     NODECK,FLAGW,STMT,SIZE=0090112,LINECNT=055,OPT=02,SORMGIN=(002,072,001),NOEXTDIC,
*OPTIONS IN EFFECT*     NEST,OPLIST
```

(2)

```
       P1:     PROC OPTIONS(MAIN);                                                PAGE     2

COMPILE-TIME MACRO PROCESSOR
MACRO SOURCE2 LISTING

  1    P1:     PROC OPTIONS(MAIN);
  2            %DCL AA CHAR, (V1,V2,V3,V4) FIXED;
  3            %AA='IF ANS>COMP THEN WORDS=''GREATER THAN'';
  4                ELSE IF ANS<COMP THEN WORDS=''LESS THAN'';
  5                     ELSE WORDS=''EQUAL TO'';
  6                PUT SKIP DATA(ANS,COMP);
  7                PUT SKIP EDIT(''ANSWER '',WORDS,'' COMPARATOR'')(A);';
  8            %V1=1; %V3=2; %V4=4;
  9            DCL (ARG(2,2) STATIC INIT(V1,V2,V3,V4), (COMP,ANS) EXT)
 10                  FIXED DEC(2,1),
 11                WORDS CHAR(12) VAR,
 12                (P2,P3) ENTRY EXT;
 13            DO I=1,2;
 14              DO J=1,2;
 15                COMP=ARG(I,J)/2;
 16                CALL P2;
 17                AA
 18                CALL P3;
 19                AA
 20            END P1;
```

(3)

```
P1:     PROC OPTIONS(MAIN);                                                    PAGE     3
```

(4)

```
   MACRO DIAGNOSTIC MESSAGES


SEVERE ERRORS.

    IEM4504I   VARIABLE V2 IS USED IN LINE NUMBER 9 BEFORE IT IS INITIALIZED. IT HAS BEEN GIVEN A NULL STRING OR
               ZERO VALUE.


  END OF MACRO  DIAGNOSTIC MESSAGES.
```

```
      P1:     PROC OPTIONS(MAIN);                                             PAGE     4
```

(5)

```
               SOURCE LISTING.


STMT LEVEL NEST
  1                P1:     PROC OPTIONS(MAIN);                                    1
  2     1                  DCL (ARG(2,2) STATIC INIT(        1 ,        0 ,        2 ,        9 1
                          4 ), (COMP,ANS) EXT)                                    9 1
                                FIXED DEC(2,1),                                  10
                             WORDS CHAR(12) VAR,                                 11
                             (P2,P3) ENTRY EXT;                                  12
  3     1                  DO I=1,2;                                            13
  4     1     1              DO J=1,2;                                          14
  5     1     2                COMP=ARG(I,J)/2;                                 15
  6     1     2                CALL P2;                                         16
  7     1     2                 IF ANS>COMP THEN WORDS='GREATER THAN';          17 1
  9     1     2              ELSE IF ANS<COMP THEN WORDS='LESS THAN';           17 1
 11     1     2                 ELSE WORDS='EQUAL TO';                          17 1
 12     1     2           PUT SKIP DATA(ANS,COMP);                              17 1
 13     1     2           PUT SKIP EDIT('ANSWER ',WORDS,' COMPARATOR')(A);       17 1
 14     1     2                 CALL P3;                                        18
 15     1     2                  IF ANS>COMP THEN WORDS='GREATER THAN';         19 1
 17     1     2              ELSE IF ANS<COMP THEN WORDS='LESS THAN';           19 1
 19     1     2                 ELSE WORDS='EQUAL TO';                          19 1
 20     1     2           PUT SKIP DATA(ANS,COMP);                              19 1
 21     1     2           PUT SKIP EDIT('ANSWER ',WORDS,' COMPARATOR')(A);       19 1
 22     1     2             END P1;                                             20
```

```
P1:     PROC OPTIONS(MAIN);                                                    PAGE    5
```

(6)

ATTRIBUTE AND CROSS-REFERENCE TABLE

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
2	ANS	STATIC,EXTERNAL,ALIGNED,DECIMAL,FIXED(2,1) 7,9,12,15,17,20
2	ARG	(2,2)STATIC,ALIGNED,INITIAL,DECIMAL,FIXED(2,1) 5
2	COMP	STATIC,EXTERNAL,ALIGNED,DECIMAL,FIXED(2,1) 5,7,9,12,15,17,20
*********	I	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 3,5
*********	J	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 4,5
1	P1	ENTRY,DECIMAL,FLOAT(SINGLE)
2	P2	EXTERNAL,ENTRY,DECIMAL,FLCAT(SINGLE) 6
2	P3	EXTERNAL,ENTRY,DECIMAL,FLOAT(SINGLE) 14
	SYSPRINT	FILE,EXTERNAL 12,13,20,21
2	WORDS	AUTOMATIC,UNALIGNED,STRING(12),CHARACTER,VARYING 8,10,11,13,16,18,19,21

```
P1:     PROC OPTIONS(MAIN);                                                    PAGE    6
```

(7)

AGGREGATE LENGTH TABLE

STATEMENT NO.	IDENTIFIER	LENGTH IN BYTES
2	ARG	8

```
P1:     PROC OPTIONS(MAIN);                                                    PAGE    7
```

(8)

STORAGE REQUIREMENTS.

THE STORAGE AREA FOR THE PROCEDURE LABELLED P1 IS 228 BYTES LONG.

THE PROGRAM CSECT IS NAMED P1   AND IS 776 BYTES LONG.

THE STATIC CSECT IS NAMED *****P1A AND IS 344 BYTES LONG.

```
P1:     PROC OPTIONS(MAIN);                                                    PAGE    8

                        EXTERNAL SYMBOL DICTIONARY
          SYMBOL          TYPE     ID      ADDR     LENGTH
          P1              SD       0001    000000   000308
          *****P1A        SD       0002    000000   000158
          IHEQINV         PR       0003    000000   000004
          IHESADA         ER       0004    000000
          IHESADB         ER       0005    000000
          IHEQERR         PR       0006    000000   000004
          IHEQTIC         PR       0007    000000   000004
          IHEMAIN         SD       0008    000000   000004
          IHENTRY         SD       0009    000000   00000C
          IHESAPC         ER       000A    000000
          *****P1B        PR       000B    000000   000004
          IHEDDOC         ER       000C    000000
          IHEDDOA         ER       000D    000000
          IHEDOBB         ER       000E    000000
          IHEDOBB         ER       000F    000000
          IHEIOBT         ER       0010    000000
          IHEIOBC         ER       0011    000000
          IHESAFA         ER       0012    000000
          COMP            CM       0013    000000   000002
          ANS             CM       0014    000000   000002
          P2              ER       0015    000000
          P3              ER       0016    000000
          IHESPRT         SD       0017    000000   000038
          IHEQSPR         PR       0018    000000   000004
          IHEDNC          ER       0019    000000
          IHEVPF          ER       001A    000000
          IHEDMA          ER       001B    000000
          IHEVPB          ER       001C    000000
          IHEVSC          ER       001D    000000
          IHEVQC          ER       001E    000000

*STATISTICS*    MACRO RECORDS =      20,SOURCE RECORDS =      22,PROG TEXT STMNTS =      22,OBJECT BYTES =      776
```

```
P1:     PROC OPTIONS(MAIN);                                                    PAGE    9 (10)

                        STATIC INTERNAL STORAGE MAP
00004C  00000000            A..   P1
000050  00000000            A..   IHEDDCC
000054  00000000            A..   IHEDDCA
000058  00000000            A..   IHEDDBB
00005C  00000000            A..   IHEDDBB
000060  00000000            A..   IHEIOBT
000064  00000000            A..   IHEIOBC
000030  00000000            A..   IHESAFA
000068  00000000            A..   COMP
00006C  00000000            A..   ANS
000070  00000000            A..   P2
000074  00000000            A..   P3
000078  00000000            A..   IHESPRT
000080  0000000200000001    CONSTANTS
000088  0000009000E000B
000090  40C3D6D4D7C1D9C1
000098  E3D6D9E7000000A4
0000A0  00070007C1D5E2E6
0000A8  C5D940C5D8E4C1D3
0000B0  40E3D6D3C5E2E240
0000B8  E3C8C1D5C7D9C5C1
0000C0  E3C5D940E3C8C1D5
0000C8  2C
0000CC  000000AC000C0000    D.V. SKELETON
0000D4  0000013C            ARGUMENT LIST
0000D8  80
0000D9  000120
0000DC  0000013C            ARGUMENT LIST
0000E0  80
0000E1  000120
0000E4  00000000            ARGUMENT LIST
0000E8  000002DC
0000EC  80
0000ED  000084
0000F0  00000000            ARGUMENT LIST
0000F4  0000028E
0000F8  80
0000F9  000084
0000FC  00000000            ARGUMENT LIST
000100  000001DE
000104  80
000105  000084
000108  00000000            ARGUMENT LIST
00010C  00000190
000110  80
000111  000084
000120  0000013C            SYM TAB
000124  04C3D6D4D7404040
00012C  0000013800000000
000134  00000000
000138  880281
00013C  0C000000            SYM TAB
000140  03C1D5E2
000144  0000015000000000
00014C  0C000000
000150  880281
000153  2C                  DED
000154  3C                  DED
000155  8A0600              DED
000038  00000112            DV..  ARG
00003C  0000000400000002
000044  00020001
000048  00020001
000118  010C                      ARG
00011A  000C
00011C  020C
00011E  040C
```

```
P1:     PROC OPTIONS(MAIN);                                                    PAGE    10

 OBJECT LISTING

* STATEMENT NUMBER  1
* PROCEDURE                                    P1
* REAL ENTRY                                   P1
000000  47 F0 F 010                  B     16(0,15)
000004                               DC    AL1(2)
000005                               DC    C'P1'
000008  000000E4                     DC    F'228'
00000C  00000000                     DC    A(SI.)
000010  90 EB D 00C                  STM   14,11,12(13)
000014  58 B0 F 00C                  L     11,12(0,15)
000018  58 00 F 008                  L     0,8(0,15)
00001C  58 F0 B 020                  L     15,32(0,11)
000020  05 EF                        BALR  14,15
000022  05 A0                        BALR  10,0
* PROLOGUE BASE
000024  41 90 D 088                  LA    9,184(0,13)
000028                       CL.22   EQU   *
000028  50 DC 0 000                  ST    13,PR..P1(12)
00002C  92 00 D 062                  MVI   98(13),X'00'
000030  92 01 D 063                  MVI   99(13),X'01'
000034  92 C0 D 000                  MVI   0(13),X'C0'
000038  D2 07 D 0A0 B 0CC            MVC   DV..WORDS(8),SKDV.
                                             .08BC
00003E  41 F0 D 0AC                  LA    15,WORDS
000042  50 F0 D 0A0                  ST    15,DV..WORDS
000046  41 A0 A 028                  LA    10,CL.20
00004A  07 00                        NOPR  0
* PROCEDURE BASE
00004C                       CL.20   EQU   *
* APPARENT ENTRY                               P1
* STATEMENT NUMBER  3
00004C  92 03 D 063                  MVI   99(13),X'03'
000050  41 80 A 016                  LA    8,CL.2
000054  D2 01 D 0A8 B 086            MVC   I(2),C..0848+2
00005A  50 80 9 020                  ST    8,32(0,9)
00005E  47 F0 A 028                  B     CL.1
000062                       CL.2    EQU   *
000062  92 03 D 063                  MVI   99(13),X'03'
000066  41 80 A 2B2                  LA    8,CL.3
00006A  D2 01 D 0A8 B 082            MVC   I(2),C..0858+2
00007C  50 8C 9 020                  ST    8,32(0,9)
000074                       CL.1    EQU   *
000074  92 03 D 063                  MVI   99(13),X'03'
* STATEMENT NUMBER  4
000078  92 04 D 063                  MVI   99(13),X'04'
00007C  48 80 D 0A8                  LH    8,I
000080  1A 88                        AR    8,8
000082  1A 88                        AR    8,8
000084  50 80 9 024                  ST    8,36(0,9)
000088  41 70 A 04E                  LA    7,CL.5
00008C  D2 01 D 0AA B 086            MVC   J(2),C..0848+2
000092  50 70 9 028                  ST    7,40(0,9)
000096  47 F0 A 060                  B     CL.4
00009A                       CL.5    EQU   *
00009A  92 04 D 063                  MVI   99(13),X'04'
00009E  41 70 A 2A4                  LA    7,CL.6
0000A2  D2 01 D 0AA B 082            MVC   J(2),C..0858+2
0000A8  50 70 9 028                  ST    7,40(0,9)
0000AC                       CL.4    EQU   *
0000AC  92 04 D 063                  MVI   99(13),X'04'
* STATEMENT NUMBER  5
0000B0  92 05 D 063                  MVI   99(13),X'05'
0000B4  18 78                        LR    7,8
0000B6  48 60 D 0AA                  LH    6,J
0000BA  1A 66                        AR    6,6
0000BC  1A 76                        AR    7,6
0000BE  41 67 B 112                  LA    6,VO..ARG(7)
0000C2  F1 21 D 090 6 000            MVO   WS1.1(3),0(2,6)
0000C8  D7 06 D 092 D 092            XC    WS1.1+2(7),WS1.1+2
0000CE  D1 00 D 098 6 001            MVN   WS1.1+8(1),1(6)
0000D4  FD 80 D 090 B 0C8            DP    WS1.1(9),C..06B8(1
                                             )
0000DA  58 70 B 068                  L     7,A..COMP
0000DE  F1 10 7 000 D 090            MVO   0(2,7),WS1.1(1)
0000E4  D1 00 7 001 D 097            MVN   1(1,7),WS1.1+7
* STATEMENT NUMBER  6
0000EA  92 06 D 063                  MVI   99(13),X'06'
0000EE  41 10 D 090                  LA    1,WS1.1
0000F2  58 F0 B 070                  L     15,A..P2
0000F6  05 EF                        BALR  14,15
* STATEMENT NUMBER  7
0000F8  92 07 D 063                  MVI   99(13),X'07'
0000FC  58 80 B 06C                  L     8,A..ANS
000100  58 70 B 068                  L     7,A..COMP
000104  F9 11 8 000 7 000            CP    0(2,8),0(2,7)
00010A  58 80 9 024                  L     8,36(0,9)
00010E  47 C0 A 0E0                  BC    12,CL.7
* STATEMENT NUMBER  8
000112  92 08 D 063                  MVI   99(13),X'08'
```

```
000116  41 60 0 00C              LA    6,12(0,0)
00011A  40 60 D 0A6              STH   6,DV..WORDS+6
00011E  58 60 D 0A0              L     6,DV..WORDS
000122  D2 0B 6 000 B 0BC        MVC   0(12,6),C..0534
000128  47 F0 A 126              B     CL.8
00012C                   CL.7    EQU   *

* STATEMENT NUMBER  9
00012C  92 09 D 063              MVI   99(13),X'09'
000130  58 70 B 06C              L     7,A..ANS
000134  58 60 B 068              L     6,A..COMP
000138  F9 11 7 000 6 000        CP    0(2,7),0(2,6)
00013E  47 A0 A 110              BC    10,CL.9

* STATEMENT NUMBER  10
000142  92 0A D 063              MVI   99(13),X'0A'
000146  41 50 0 009              LA    5,9(0,0)
00014A  40 50 D 0A6              STH   5,DV..WORDS+6
00014E  58 50 D 0A0              L     5,DV..WORDS
000152  D2 08 5 000 B 0B3        MVC   0(9,5),C..0550
000158  47 F0 A 126              B     CL.8
00015C                   CL.9    EQU   *

* STATEMENT NUMBER  11
00015C  92 0B D 063              MVI   99(13),X'0B'
000160  41 70 0 008              LA    7,8(0,0)
000164  40 70 D 0A6              STH   7,DV..WORDS+6
000168  58 70 D 0A0              L     7,DV..WORDS
00016C  D2 07 7 000 B 0AB        MVC   0(8,7),C..0568
000172                   CL.8    EQU   *

* STATEMENT NUMBER  12
000172  92 0C D 063              MVI   99(13),X'0C'
000176  41 10 B 108              LA    1,SKPL..06FC
00017A  58 F0 B 064              L     15,A..IHEIOBC
00017E  05 EF                    BALR  14,15
000180  41 10 B 0DC              LA    1,SKPL..0800
000184  58 F0 B 054              L     15,A..IHEDDOA
000188  05 EF                    BALR  14,15
00018A  58 F0 B 050              L     15,A..IHEDOOC
00018E  05 EF                    BALR  14,15
                         CL.23   EQU   *
000190  92 0C D 063              MVI   99(13),X'0C'
000194  58 F0 B 060              L     15,A..IHEIOBT
000198  05 EF                    BALR  14,15

* STATEMENT NUMBER  13
00019A  92 0D D 063              MVI   99(13),X'0D'
00019E  41 10 B 0FC              LA    1,SKPL..0748
0001A2  58 F0 B 064              L     15,A..IHEIOBC
0001A6  05 EF                    BALR  14,15
0001A8  41 80 A 182              LA    8,CL.16
0001AC  41 10 B 09C              LA    1,DV..C..059C
0001B0  41 20 B 153              LA    2,DED..C..059C
0001B4  05 78                    BALR  7,8
0001B6  41 10 D 0A0              LA    1,DV..WORDS
0001BA  41 20 B 154              LA    2,DED..WORDS
0001BE  05 78                    BALR  7,8
0001C0  41 10 B 088              LA    1,DV..C..05B4
0001C4  41 20 B 153              LA    2,DED..C..05B4
0001C8  05 78                    BALR  7,8
0001CA  47 F0 A 18E              B     CL.17
0001CE                   CL.16   EQU   *
0001CE  58 F0 B 05C              L     15,A..IHEDOBB
0001D2  18 E7                    LR    14,7
0001D4  05 8F                    BALR  8,15
0001D6  47 F0 A 182              B     CL.16
0001DA                   CL.17   EQU   *
0001DA  58 80 9 024              L     8,36(0,9)
                         CL.24   EQU   *
0001DE  92 0D D 063              MVI   99(13),X'0D'
0001E2  58 F0 B 060              L     15,A..IHEIOBT
0001E6  05 EF                    BALR  14,15

* STATEMENT NUMBER  14
0001E8  92 0E D 063              MVI   99(13),X'0E'
0001EC  41 10 D 090              LA    1,WS1.1
0001F0  58 F0 B 074              L     15,A..P3
0001F4  05 EF                    BALR  14,15

* STATEMENT NUMBER  15
0001F6  92 0F D 063              MVI   99(13),X'0F'
0001FA  58 80 B 06C              L     8,A..ANS
0001FE  58 70 B 068              L     7,A..COMP
000202  F9 11 8 000 7 000        CP    0(2,8),0(2,7)
000208  58 80 9 024              L     8,36(0,9)
00020C  47 C0 A 1DE              BC    12,CL.10

* STATEMENT NUMBER  16
000210  92 10 D 063              MVI   99(13),X'10'
000214  41 60 0 00C              LA    6,12(0,0)
000218  40 60 D 0A6              STH   6,DV..WORDS+6
00021C  58 60 D 0A0              L     6,DV..WORDS
000220  D2 0B 6 000 B 0BC        MVC   0(12,6),C..0534
000226  47 F0 A 224              B     CL.11
00022A                   CL.10   EQU   *

* STATEMENT NUMBER  17
00022A  92 11 D 063              MVI   99(13),X'11'
00022E  58 70 B 06C              L     7,A..ANS
000232  58 60 B 068              L     6,A..COMP
000236  F9 11 7 000 6 000        CP    0(2,7),0(2,6)
```

```
P1:     PROC OPTIONS(MAIN);                                                    PAGE    12 (13)

00023C  47 A0 A 20E                BC    10,CL.12
* STATEMENT NUMBER  18
000240  92 12 D 063                MVI   99(13),X'12'
000244  41 50 0 009                LA    5,9(0,0)
000248  40 50 D 0A6                STH   5,DV..WORDS+6
00024C  58 50 D 0A0                L     5,DV..WORDS
000250  D2 08 5 000 B 0B3          MVC   0(9,5),C..0550
000256  47 F0 A 224                B     CL.11
00025A                     CL.12   EQU   *
* STATEMENT NUMBER  19
00025A  92 13 D 063                MVI   99(13),X'13'
00025E  41 70 0 008                LA    7,8(0,0)
000262  40 70 D 0A6                STH   7,DV..WCRDS+6
000266  58 70 D 0A0                L     7,DV..WORDS
00026A  D2 07 7 000 B 0AB          MVC   0(8,7),C..0568
000270                     CL.11   EQU   *
* STATEMENT NUMBER  20
000270  92 14 D 063                MVI   99(13),X'14'
000274  41 10 B 0F0                LA    1,SKPL..0774
000278  58 FC B 064                L     15,A..IHEIOBC
00027C  05 EF                      BALR  14,15
00027E  41 10 B 0C4                LA    1,SKPL..0834
000282  58 F0 B 054                L     15,A..IHEDDOA
000286  05 EF                      BALR  14,15
000288  58 F0 B 050                L     15,A..IHEDDOC
00028C  05 EF                      BALR  14,15
                           CL.25   EQU   *
00028E  92 14 D 063                MVI   99(13),X'14'
000292  58 F0 B 060                L     15,A..IHEIOBT
000296  05 EF                      BALR  14,15
* STATEMENT NUMBER  21
000298  92 15 D 063                MVI   99(13),X'15'
00029C  41 10 B 0E4                LA    1,SKPL..07A0
0002A0  58 FC B 064                L     15,A..IHEIOBC
0002A4  05 EF                      BALR  14,15
0002A6  41 80 A 280                LA    8,CL.18
0002AA  41 10 B 09C                LA    1,DV..C..059C
0002AE  41 20 B 153                LA    2,DED..C..059C
0002B2  05 78                      BALR  7,8
0002B4  41 10 D 0A0                LA    1,DV..WORDS
0002B8  41 20 B 154                LA    2,DED..WORDS
0002BC  05 78                      BALR  7,8
0002BE  41 10 B 088                LA    1,DV..C..05B4
0002C2  41 20 B 153                LA    2,DED..C..05B4
0002C6  05 78                      BALR  7,8
0002C8  47 F0 A 28C                B     CL.19
0002CC                     CL.18   EQU   *
0002CC  58 F0 B 05C                L     15,A..IHEDOBB
0002D0  18 E7                      LR    14,7
0002D2  05 8F                      BALR  8,15
0002D4  47 F0 A 280                B     CL.18
0002D8                     CL.19   EQU   *
0002D8  58 80 9 024                L     8,36(0,9)
                           CL.26   EQU   *
0002DC  92 15 D 063                MVI   99(13),X'15'
0002E0  58 F0 B 060                L     15,A..IHEIOBT
0002E4  05 EF                      BALR  14,15
* STATEMENT NUMBER  22
0002E6  92 16 D 063                MVI   99(13),X'16'
0002EA  58 70 9 028                L     7,40(0,9)
0002EE  07 F7                      BR    7
0002F0                     CL.6    EQU   *
0002F0  92 16 D 063                MVI   99(13),X'16'
* STATEMENT NUMBER  23
0002F4  92 17 D 063                MVI   99(13),X'17'
0002F8  58 80 9 020                L     8,32(0,9)
0002FC  07 F8                      BR    8
0002FE                     CL.3    EQU   *
* STATEMENT NUMBER  24
0002FE  92 18 D 063                MVI   99(13),X'18'
000302  58 F0 B 030                L     15,A..IHESAFA
000306  05 EF                      BALR  14,15
* END PROCEDURE                           P1
                                   END
```

```
P1:     PROC OPTIONS(MAIN);                                                    PAGE     13
```

(14)

```
   COMPILER DIAGNOSTICS.


WARNINGS.

   IEM0227I          NO FILE/STRING OPTION SPECIFIED IN ONE OR MORE GET/PUT STATEMENTS. SYSIN/SYSPRINT HAS BEEN

                     ASSUMED IN EACH CASE.

   IEM0764I          ONE OR MORE FIXED BINARY ITEMS OF PRECISION 15 OR LESS HAVE BEEN GIVEN HALFWORD STORAGE. THEY

                     ARE FLAGGED '*********' IN THE XREF/ATR LIST.


 END OF DIAGNOSTICS.

AUXILIARY STORAGE WILL NOT BE USED FOR DICTIONARY WHEN SIZE =  65K

COMPILE TIME         .05 MINS
ELAPSED TIME         .68 MINS
```

```
VERSION 5.0                                OS/360 PL/I COMPILER (F)                PAGE     14
                                                                                   DATE 69.273
```

(15)

```
PL/I F COMPILER OPTIONS SPECIFIED ARE AS FOLLOWS--
     E,NT

THE COMPLETE LIST OF OPTIONS USED DURING THIS COMPILATION IS--   EBCDIC
                                                                CHAR60
                                                              NOMACRO
                                                                SOURCE2
                                                              NOMACDCK
                                                                COMP
                                                                SOURCE
                                                              NOATR
                                                              NOXREF
                                                                EXTREF
                                                              NOLIST
                                                                LOAD
                                                              NODECK
                                                                FLAGW
                                                                STMT
                                                                SIZE=0090112
                                                                LINECNT=055
                                                                OPT=02
                                                                SORMGIN=(002,072,001)
                                                              NOEXTDIC
                                                                NEST
                                                                OPLIST


*OPTIONS IN EFFECT*     EBCDIC,CHAR60,NOMACRO,SOURCE2,NOMACDCK,COMP,SOURCE,NOATR,NOXREF,EXTREF,NOLIST,LOAD,
*OPTIONS IN EFFECT*     NODECK,FLAGW,STMT,SIZE=0090112,LINECNT=055,OPT=02,SORMGIN=(002,072,001),NOEXTDIC,
*OPTIONS IN EFFECT*     NEST,OPLIST
```

```
        P2:     PROC;                                                    PAGE     15

STMT LEVEL NEST
  1                P2:     PROC;
  2     1                  DCL (COMP,ANS) FIXED DEC(2,1) EXT;
  3     1                  ON ZDIV BEGIN;
  5     2                    ANS=0;
  6     2                    PUT SKIP LIST('NEXT COMMENT INVALID');
  7     2                    GO TO OUT;
  8     2                    END;
  9     1                  ANS=1/COMP;
 10     1          OUT:    END P2;
```

16

```
 P2:     PROC;                                                           PAGE     16

STORAGE REQUIREMENTS.
---------------------

THE STORAGE AREA (IN STATIC) FOR THE PROCEDURE LABELLED P2 IS 200 BYTES LONG.

THE STORAGE AREA FOR THE ON-UNIT AT STATEMENT NO. 4 IS 176 BYTES LONG.

THE PROGRAM CSECT IS NAMED P2        AND IS 442 BYTES LONG.

THE STATIC  CSECT IS NAMED *****P2A AND IS 368 BYTES LONG.
```

17

```
        P2:     PROC;                                                    PAGE     17

                        EXTERNAL SYMBOL DICTIONARY
                 SYMBOL          TYPE     ID      ADDR     LENGTH
                 P2               SD     0001    000000   0001BA
                 *****P2A         SD     0002    000000   000170
                 IHEQINV          PR     0003    000000   000004
                 IHESADA          ER     0004    000000
                 IHESADB          ER     0005    000000
                 IHEQERR          PR     0006    000000   000004
                 IHEQTIC          PR     0007    000000   000004
                 IHENTRY          SD     0008    000000   00000C
                 IHESAPC          ER     0009    000000
                 IHEQLWF          PR     000A    000000   000004
                 IHEQSLA          PR     000B    000000   000004
                 IHEQLWO          PR     000C    000000   000004
                 *****P2B         PR     000D    000000   000004
                 *****P2C         PR     000E    000000   000004
                 IHELDOB          ER     000F    000000
                 IHEIOBT          ER     0010    0000C0
                 IHEIOBC          ER     0011    000000
                 IHESAFA          ER     0012    000000
                 IHESAFC          ER     0013    000000
                 ANS              CM     0014    000000   000002
                 COMP             CM     0015    000000   000002
                 IHESPRT          SD     0016    000000   000038
                 IHEQSPR          PR     0017    000000   000004
                 IHEVSC           ER     0018    000000

*STATISTICS*     SOURCE RECORDS =        9,PROG TEXT STMNTS =       10,OBJECT BYTES =      442
```

18

```
          P2:     PROC;                                                                    PAGE     18 (19)

    COMPILER DIAGNOSTICS.


 WARNINGS.

    IEM0227I          NO FILE/STRING OPTION SPECIFIED IN ONE OR MORE GET/PUT STATEMENTS. SYSIN/SYSPRINT HAS BEEN

                      ASSUMED IN EACH CASE.

    IEM0526I      1   OPTION MAIN HAS NOT BEEN SPECIFIED FOR THE EXTERNAL PROCEDURE, STATEMENT NUMBER 1


 END OF DIAGNOSTICS.

AUXILIARY STORAGE WILL NOT BE USED FOR DICTIONARY WHEN SIZE =  65K

COMPILE TIME          .02 MINS

ELAPSED TIME          .49 MINS
```

```
VERSION 5.0                                  OS/360 PL/I COMPILER (F)                  PAGE     19 (20)
                                                                                       DATE 69.273

PL/I F COMPILER OPTIONS SPECIFIED ARE AS FOLLOWS--
     E

THE COMPLETE LIST OF OPTIONS USED DURING THIS COMPILATION IS--   EBCDIC
                                                                 CHAR60
                                                               NOMACRO
                                                                 SOURCE2
                                                               NOMACDCK
                                                                 COMP
                                                                 SOURCE
                                                               NOATR
                                                               NOXREF
                                                                 EXTREF
                                                               NOLIST
                                                                 LOAD
                                                               NODECK
                                                                 FLAGW
                                                                 STMT
                                                                 SIZE=0090112
                                                                 LINECNT=055
                                                                 OPT=02
                                                                 SORMGIN=(002,072,001)
                                                               NOEXTDIC
                                                                 NEST
                                                                 OPLIST

*OPTIONS IN EFFECT*     EBCDIC,CHAR60,NOMACRO,SOURCE2,NOMACDCK,COMP,SOURCE,NOATR,NOXREF,EXTREF,NOLIST,LOAD,
*OPTIONS IN EFFECT*     NODECK,FLAGW,STMT,SIZE=0090112,LINECNT=055,OPT=02,SORMGIN=(002,072,001),NOEXTDIC,
*OPTIONS IN EFFECT*     NEST,OPLIST
```

```
        P3:     PROC;                                                    PAGE     20

STMT LEVEL NEST
  1              P3:     PROC;
  2     1                DCL (COMP,ANS) FIXED DEC(2,1) EXT;
  3     1                ANS=COMP*COMP;
  4     1                END P3;
```

(21)

```
 P3:     PROC;                                                           PAGE     21

STORAGE REQUIREMENTS.
---------------------

THE STORAGE AREA (IN STATIC) FOR THE PROCEDURE LABELLED P3 IS 184 BYTES LONG.

THE PROGRAM CSECT IS NAMED P3       AND IS 246 BYTES LONG.

THE STATIC  CSECT IS NAMED *****P3A AND IS 264 BYTES LONG.
```

(22)

```
          P3:     PROC;                                                  PAGE     22

                        EXTERNAL SYMBOL DICTIONARY
                  SYMBOL          TYPE     ID      ADDR      LENGTH
                  P3              SD       0001    000000    0000F6
                  *****P3A        SD       0002    000000    000108
                  IHEQINV         PR       0003    000000    000004
                  IHESADA         ER       0004    000000
                  IHESADB         ER       0005    000000
                  IHEQERR         PR       0006    000000    000004
                  IHEQTIC         PR       0007    000000    000004
                  IHENTRY         SD       0008    000000    00000C
                  IHESAPC         ER       0009    000000
                  IHEQLWF         PR       000A    000000    000004
                  IHEQSLA         PR       000B    000000    000004
                  IHEQLWO         PR       000C    000000    000004
                  *****P3B        PR       000D    000000    000004
                  IHESAFA         ER       000E    000000
                  ANS             CM       000F    000000    000002
                  COMP            CM       0010    000000    000002

*STATISTICS*     SOURCE RECORDS =        4,PROG TEXT STMNTS =        4,OBJECT BYTES =      246
```

(23)

```
          P3:     PROC;                                                                    PAGE     23
```

(24)

```
    COMPILER DIAGNOSTICS.


 WARNINGS.

    IEM0526I      1   OPTION MAIN HAS NOT BEEN SPECIFIED FOR THE EXTERNAL PROCEDURE, STATEMENT NUMBER 1



 END OF DIAGNOSTICS.

AUXILIARY STORAGE WILL NOT BE USED FOR DICTIONARY WHEN SIZE =  65K

COMPILE TIME          .02 MINS

ELAPSED TIME          .33 MINS
```

(25)

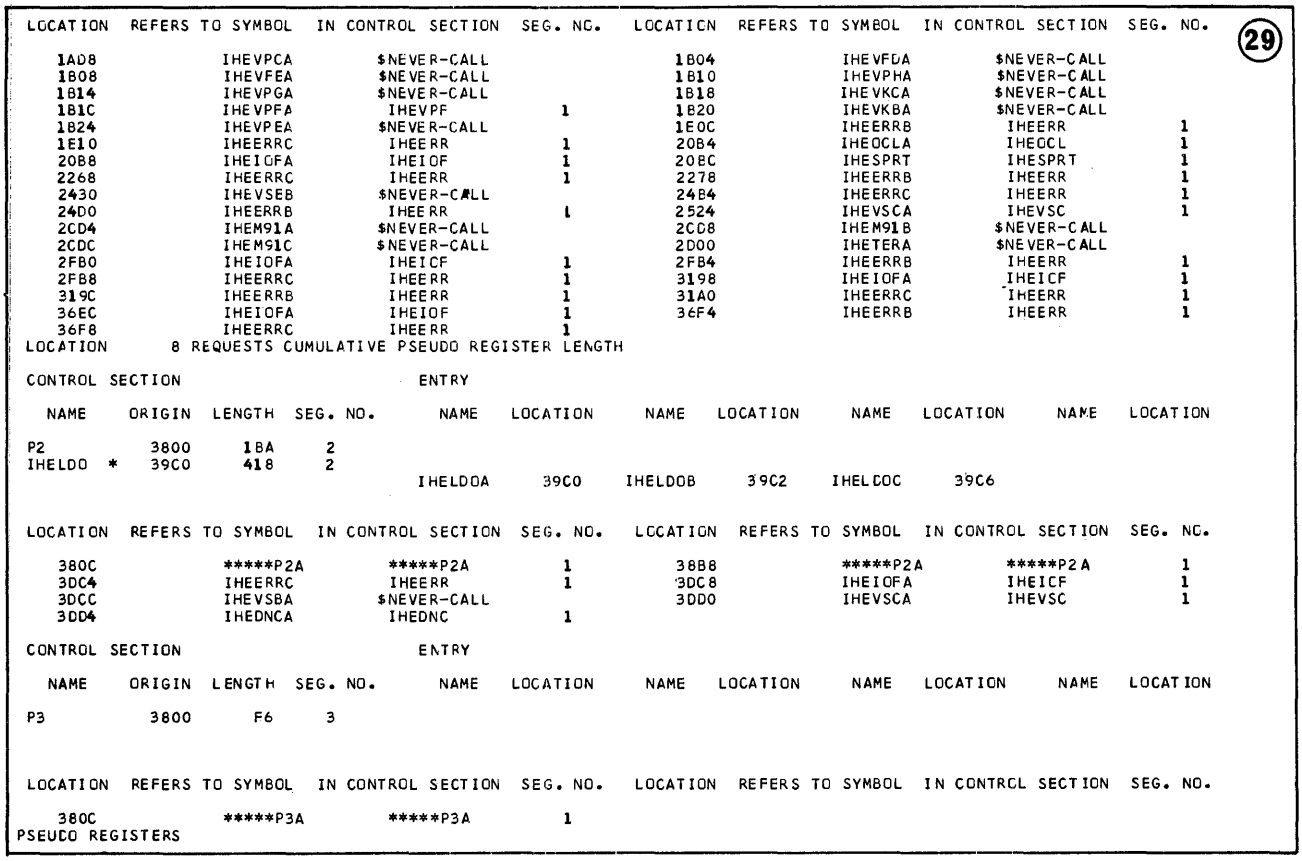
```
IEF285I   SYS69273.T115427.SV000.J063PGEX.R0000009      SYSOUT
IEF285I   VOL SER NOS= M65294.
IEF285I   SYS69273.T115427.RV000.J063PGEX.LCADSET       PASSED
IEF285I   VOL SER NOS= 231422.
IEF285I   SYS69273.T115427.RV000.J063PGEX.R0000010      DELETED
IEF285I   VOL SER NOS= 231423.
IEF285I   SYS69273.T115427.RV000.J063PGEX.R0000011      DELETED
IEF285I   VOL SER NOS= 231424.
IEF285I   SYS69273.T115427.RV000.J063PGEX.S0000012      SYSIN
IEF285I   VOL SER NOS= M65295.
IEF285I   SYS69273.T115427.RV000.J063PGEX.S0000012      DELETED
IEF285I   VOL SER NOS= M65295.
  END OF STEP 'PL1L    '   JOB 'J063PGEX'    STEPTIME=00.00.07  CLOCK=11.56.49   DATE=69.273   RETURN CODE=0012
XXLKED   EXEC  PGM=IEWL,PARM='MAP,LIST',COND=(16,EQ,PL1L),              *00160000
XX             REGION=108K                                               00180000
XXSYSLIB   DD  DSNAME=SYS1.PL1LIB,DISP=SHR                               00200000
XXSYSLMOD  DD  DSNAME=&&GOSET(GO),DISP=(MOD,PASS),                       *00220000
XX             UNIT=SYSDA,SPACE=(1024,(50,20,1))                         00240000
XXSYSUT1   DD  UNIT=SYSDA,SPACE=(1024,(200,20))                          00260000

XXSYSPRINT DD  SYSOUT=A,DCB=BLKSIZE=605                                  00280000
XXSYSLIN   DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)                        00300000
XX             DD  DDNAME=SYSIN                                          00320000
//LKED.SYSIN DD *
IEF236I ALLOC. FOR J063PGEX LKED     COLEEX
IEF237I SYSLIB   ON 295
IEF237I          ON 295
IEF237I          ON 100
IEF237I SYSLMOD  ON 342
IEF237I SYSUT1   ON 343
IEF237I SYSPRINT ON 293
IEF237I SYSLIN   ON 342
IEF237I          ON 290
```

```
 F88-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED LIST,XREF,OVLY
          VARIABLE OPTIONS USED - SIZE=(104448,59392)              DEFAULT OPTION(S) USED
IEW0000      OVERLAY X
IEW0000      INSERT P2
IEW0000      INSERT  IHELDO
IEW0000      OVERLAY X
IEW0000      INSERT P3
IEW0461  IHEDDPA
IEW0461  IHEVPAA
IEW0461  IHEDDPB
IEW0461  IHEDDPC
IEW0461  IHEDDPD
IEW0461  IHEUPAB
IEW0461  IHEVSBA
IEW0461  IHEVSFA
IEW0461  IHEVSEB
IEW0461  IHEVSEA
IEW0461  IHEVPCA
IEW0461  IHEVPDA
IEW0461  IHEVPEA
IEW0461  IHEVPGA
IEW0461  IHEVPHA
IEW0461  IHEVFAA
IEW0461  IHEVFBA
IEW0461  IHEVFCA
IEW0461  IHEVFDA
IEW0461  IHEVFEA
IEW0461  IHEVKBA
IEW0461  IHEVKCA
IEW0461  IHEVKFA
IEW0461  IHEVKGA
IEW0461  IHEM91A
IEW0461  IHEM91B
IEW0461  IHEM91C
IEW0461  IHETERA

                                CROSS REFERENCE TABLE

  CONTROL SECTION                          ENTRY
    NAME     ORIGIN  LENGTH  SEG. NO.      NAME   LOCATION     NAME   LOCATION     NAME   LOCATION     NAME   LOCATION
 $SEGTAB         00      24     1
  P1             28     308     1
  *****P1A      330     158     1
  IHEMAIN       488       4     1
  IHENTRY       490       C     1
  IHESPRT       4A0      38     1
  *****P2A      4D8     170     1
  *****P3A      648     108     1
  IHEDDO  *     750     288     1
                                       IHEDDOA      750    IHEDDOB      752    IHEDDOC      754    IHEDDOD      756
                                       IHEDDOE      758
```

NAME		ORIGIN	LENGTH	SEG. NO.
IHEDNC	*	9D8	284	1
IHEDOB	*	C60	144	1
IHEIOB	*	DA8	1E4	1
IHESAP	*	F90	AB8	1
IHEDMA	*	1A48	F8	1
IHEIOF	*	1B40	2DC	1
IHEPRT	*	1E20	2C8	1
IHEVPB	*	20E8	1A2	1
IHEVPF	*	2290	50	1
IHEVQC	*	22E0	268	1
IHEVSC	*	2548	AC	1
IHEERR	*	25F8	729	1
IHEIOD	*	2D28	29A	1
IHEIOP	*	2FC8	1E8	1
IHECCL	*	31B8	554	1
IHEBEG	*	3710	80	1
IHESIZ	*	3790	C	1
IHETAB	*	37A0	C	1
COMP		37B0	2	1
ANS		37B8	2	1
$ENTAB		37C0	3C	1

NAME	LOCATION	NAME	LOCATION	NAME	LOCATION	NAME	LOCATION
IHEDNCA	9D8						
IHEDOBA	C60	IHEDOBB	C62	IHEDOBC	C64		
IHEIOBA	DA8	IHEIOBB	DB0	IHEIOBC	DB8	IHEICBD	DC0
IHEIOBE	DC8	IHEIOBT	EB4				
IHESADA	F90	IHESAPC	FAA	IHESAPD	FB2	IHESAPA	FBA
IHESAPB	FC2	IHESADF	FCA	IHESADB	FD2	IHESADE	FDA
IHESAFC	FE2	IHESAFA	FEA	IHESAFB	FF2	IHESAFD	FFA
IHESARA	1002	IHESAFQ	100A	IHESARC	17C6	IHESADD	18B4
IHESAFF	18EE						
IHEDMAA	1A48						
IHEIOFB	1B40	IHEIOFA	1B42	IHEITAZ	1DDE	IHEITAX	1DEA
IHEITAA	1DFE						
IHEPRTA	1E20	IHEPRTB	1E22				
IHEVPBA	20E8						
IHEVPFA	2290						
IHEVQCA	22E0						
IHEVSCA	2548						
IHEERRD	25F8	IHEERRC	2602	IHEERRB	260C	IHEERRA	2616
IHEERRE	2C8E						
IHEIODG	2D28	IHEIODP	2D2A	IHEIODT	2E22		
IHEIOPA	2FC8	IHEIOPB	2FCA	IHEIOPC	2FCE		
IHECCLA	31B8	IHEOCLB	31BA	IHEOCLC	31BC	IHEOCLD	31BE
IHEBEGN	3710	IHEBEGA	3750				
IHESIZE	3790						
IHETABS	37A0						

LOCATION	REFERS TO SYMBOL	IN CONTROL SECTION	SEG. NO.	LOCATION	REFERS TC SYMBOL	IN CCNTROL SECTION	SEG. NO.
34	*****P1A	*****P1A	1	350	IHESADA	IHESAP	1
354	IHESADB	IHESAP	1	358	P1	P1	1
35C	P1	P1	1	37C	P1	P1	1
380	IHEDDOC	IHEDCO	1	384	IHEDDOA	IHEDDO	1
388	IHEDOBB	IHEDOB	1	38C	IHEDOBB	IHEDCB	1
390	IHEIOBT	IHEICB	1	394	IHEIOBC	IHEICB	1
360	IHESAFA	IHESAP	1	398	COMP	COMP	1
39C	ANS	ANS	1	3A0	P2	P2	2
3A4	P3	P3	3	3A8	IHESPRT	IHESPRT	1
414	IHESPRT	IHESPRT	1	418	P1	P1	1
420	IHESPRT	IHESPRT	1	424	P1	P1	1
42C	IHESPRT	IHESPRT	1	430	P1	P1	1
438	IHESPRT	IHESPRT	1	43C	P1	P1	1
461	COMP	COMP	1	479	ANS	ANS	1
488	P1	P1	1	498	IHESAPC	IHESAP	1
4F8	IHESADA	IHESAP	1	4FC	IHESADB	IHESAP	1
500	P2	P2	2	504	P2	P2	2
51D	P2	P2	2	520	P2	P2	2
524	P2	P2	2	528	IHELDOB	IHELDO	2
52C	IHEIOBT	IHEICB	1	530	IHEIOBC	IHEIOB	1
508	IHESAFA	IHESAP	1	534	IHESAFC	IHESAP	1
538	ANS	ANS	1	53C	COMP	COMP	1
540	IHESPRT	IHESPRT	1	56C	IHESPRT	IHESPRT	1
570	P2	P2	2	668	IHESADA	IHESAP	1
66C	IHESADB	IHESAP	1	670	P3	P3	3
674	P3	P3	3	688	P3	P3	3
678	IHESAFA	IHESAP	1	68C	ANS	ANS	1
690	COMP	COMP	1	9B8	IHEIOFA	IHEIOF	1
9BC	IHELDOC	IHELDO	2	9C0	IHEPRTB	IHEPRT	1
9C4	IHEDDPA	$NEVER-CALL		9C8	IHEDDPB	$NEVER-CALL	
9CC	IHEDDPC	$NEVER-CALL		9D0	IHEDDPD	$NEVER-CALL	
C40	IHEDMAA	IHEDMA	1	C44	IHEUPAB	$NEVER-CALL	
C48	IHEVSCA	IHEVSC	1	C4C	IHEVSEB	$NEVER-CALL	
C50	IHEVQCA	IHEVQC	1	D84	IHEERRC	IHEERR	1
D88	IHEIODP	IHEIOD	1	D8C	IHEIODT	IHEIOD	1
D90	IHEDNCA	IHEDNC	1	D94	IHEVSCA	IHEVSC	1
D98	IHEVSEA	$NEVER-CALL		D9C	IHEVSBA	$NEVER-CALL	
DA0	IHEVSFA	$NEVER-CALL		F60	IHEIOPA	IHEIOP	1
F6C	IHEIOPA	IHEIOP	1	F64	IHEIOPB	IHEIOP	1
F68	IHEIOPC	IHEIOP	1	F70	IHEOCLC	IHEOCL	1
F84	IHEERRB	IHEERR	1	F88	IHEERRC	IHEERR	1
11FD	IHEERRA	IHEERR	1	194C	IHEMAIN	IHEMAIN	1
196C	IHEOCLD	IHEOCL	1	1970	IHESIZE	IHESIZ	1
1974	IHEBEGA	IHEBEG	1	19F4	IHEITAX	IHEIOF	1
19F8	IHEERRB	IHEERR	1	19FC	IHEERRC	IHEERR	1
1A00	IHETABS	IHETAB	1	1A04	IHEITAZ	IHEIOF	1
1A38	IHEPRTA	IHEPRT	1	1A3C	IHEPRTB	IHEPRT	1
1A40	IHEDDOD	IHEDDO	1	1A44	IHEOCLC	IHEOCL	1
1AB4	IHEVFBA	$NEVER-CALL		1AB8	IHEVFCA	$NEVER-CALL	
1AC0	IHEVFAA	$NEVER-CALL		1AC4	IHEVPAA	$NEVER-CALL	
1AC8	IHEVKGA	$NEVER-CALL		1ACC	IHEVPDA	$NEVER-CALL	
1AD0	IHEVKFA	$NEVER-CALL		1AD4	IHEVPBA	IHEVPB	1



LOCATION	REFERS TO SYMBOL	IN CONTROL SECTION	SEG. NO.	LOCATION	REFERS TO SYMBOL	IN CONTROL SECTION	SEG. NO.
1AD8	IHEVPCA	$NEVER-CALL		1B04	IHEVFDA	$NEVER-CALL	
1B08	IHEVFEA	$NEVER-CALL		1B10	IHEVPHA	$NEVER-CALL	
1B14	IHEVPGA	$NEVER-CALL		1B18	IHEVKCA	$NEVER-CALL	
1B1C	IHEVPFA	IHEVPF	1	1B20	IHEVKBA	$NEVER-CALL	
1B24	IHEVPEA	$NEVER-CALL		1E0C	IHEERRB	IHEERR	1
1E10	IHEERRC	IHEERR	1	20B4	IHEOCLA	IHEOCL	1
20B8	IHEIOFA	IHEIOF	1	20BC	IHESPRT	IHESPRT	1
2268	IHEERRC	IHEERR	1	2278	IHEERRB	IHEERR	1
2430	IHEVSEB	$NEVER-CALL		24B4	IHEERRC	IHEERR	1
24D0	IHEERRB	IHEERR	1	2524	IHEVSCA	IHEVSC	1
2CD4	IHEM91A	$NEVER-CALL		2CD8	IHEM91B	$NEVER-CALL	
2CDC	IHEM91C	$NEVER-CALL		2D00	IHETERA	$NEVER-CALL	
2FB0	IHEIOFA	IHEICF	1	2FB4	IHEERRB	IHEERR	1
2FB8	IHEERRC	IHEERR	1	3198	IHEIOFA	IHEICF	1
319C	IHEERRB	IHEERR	1	31A0	IHEERRC	IHEERR	1
36EC	IHEIOFA	IHEIOF	1	36F4	IHEERRB	IHEERR	1
36F8	IHEERRC	IHEERR	1				

LOCATION 8 REQUESTS CUMULATIVE PSEUDO REGISTER LENGTH

CONTROL SECTION NAME	ORIGIN	LENGTH	SEG. NO.	ENTRY NAME	LOCATION	NAME	LOCATION	NAME	LOCATION	NAME	LOCATION
P2	3800	1BA	2								
IHELDO *	39C0	418	2	IHELDOA	39C0	IHELDOB	39C2	IHELDOC	39C6		

LOCATION	REFERS TO SYMBOL	IN CONTROL SECTION	SEG. NO.	LOCATION	REFERS TO SYMBOL	IN CONTROL SECTION	SEG. NO.
380C	*****P2A	*****P2A	1	38B8	*****P2A	*****P2A	1
3DC4	IHEERRC	IHEERR	1	3DC8	IHEIOFA	IHEICF	1
3DCC	IHEVSBA	$NEVER-CALL		3DD0	IHEVSCA	IHEVSC	1
3DD4	IHEDNCA	IHEDNC	1				

CONTROL SECTION NAME	ORIGIN	LENGTH	SEG. NO.	ENTRY NAME	LOCATION	NAME	LOCATION	NAME	LOCATION	NAME	LOCATION
P3	3800	F6	3								

LOCATION	REFERS TO SYMBOL	IN CONTROL SECTION	SEG. NO.	LOCATION	REFERS TO SYMBOL	IN CONTROL SECTION	SEG. NO.
380C	*****P3A	*****P3A	1				

PSEUDO REGISTERS

```
    NAME     ORIGIN  LENGTH       NAME     ORIGIN  LENGTH       NAME     ORIGIN  LENGTH       NAME     ORIGIN  LENGTH
  IHEQINV        00       4    IHEQERR          4       4    IHEQTIC          8       4    *****P1B         C       4
  IHEQSPR        10       4    IHEQLWF         14       4    IHEQSLA         18       4    IHEQLW0         1C       4
  *****P2B       20       4    *****P2C        24       4    *****P3B        28       4    IHEQFOP         2C       4
  IHEQCFL        30       8    IHEQADC         38       4    IHEQLSA         3C       4    IHEQLW1         40       4
  IHEQLW2        44       4    IHEQLW3         48       4    IHEQLW4         4C       4    IHEQLWE         50       4
  IHEQLCA        54       4    IHEQVDA         58       4    IHEQFVD         5C       4    IHEQLPR         60       4
  IHEQSAR        64       4    IHEQRTC         68       4    IHEQSFC         6C       4    IHEQXLV         70       8
  IHEQEVT        78       8

 TOTAL LENGTH OF PSEUDO REGISTERS     ' 80
 ENTRY ADDRESS      490
 TOTAL LENGTH      3DD8

****GO        DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET

                                          DIAGNOSTIC MESSAGE DIRECTORY

  IEW0461 WARNING - SYMBOL PRINTED IS AN UNRESOLVED EXTERNAL REFERENCE, NCAL WAS SPECIFIED.
```

30

```
IEF285I   SYS1.PL1LIB                                    KEPT
IEF285I   VOL SER NOS= 2301C0.
IEF285I   SYS69273.T115427.RV000.J063PGEX.GOSET          PASSED
IEF285I   VOL SER NOS= 231422.
IEF285I   SYS69273.T115427.RV000.J063PGEX.R0000013       DELETED
IEF285I   VOL SER NOS= 231423.
IEF285I   SYS69273.T115427.SV000.J063PGEX.R0000014       SYSOUT
IEF285I   VOL SER NOS= M65293.
IEF285I   SYS69273.T115427.RV000.J063PGEX.LOADSET        DELETED
IEF285I   VOL SER NOS= 231422.
IEF285I   SYS69273.T115427.RV000.J063PGEX.SC000015       SYSIN
IEF285I   VOL SER NOS= M65295.
IEF285I   SYS69273.T115427.RV000.J063PGEX.SC000015       DELETED
IEF285I   VOL SER NOS= M65295.
   END OF STEP 'LKED    '   JOB 'J063PGEX'    STEPTIME=00.00.02  CLOCK=11.57.14   DATE=69.273   RETURN CODE=0004
XXGO     EXEC  PGM=*.LKED.SYSLMOD,COND=((9,LT,LKED),(16,EQ,PL1L)),     *00340000
XX             REGION=108K                                              00360000
XXSYSPRINT DD  SYSOUT=A                                                 00380000
IEF236I ALLOC. FOR J063PGEX GO       CCLEEX
IEF237I PGM=*.DD CN 342
IEF237I SYSPRINT ON 294
```

31

```
ANS=  2.0                COMP=  0.5;
ANSWER GREATER THAN COMPARATOR
ANS=  0.2                COMP=  0.5;
ANSWER LESS THAN COMPARATOR
NEXT COMMENT INVALID
ANS=  0.0                COMP=  0.0;
ANSWER EQUAL TO COMPARATOR
ANS=  0.0                COMP=  0.0;
ANSWER EQUAL TO COMPARATOR
ANS=  1.0                COMP=  1.0;
ANSWER EQUAL TO COMPARATOR
ANS=  1.0                COMP=  1.0;
ANSWER EQUAL TO COMPARATOR
ANS=  0.5                COMP=  2.0;
ANSWER LESS THAN COMPARATOR
ANS=  4.0                COMP=  2.0;
ANSWER GREATER THAN COMPARATOR
```

(32)

```
IEF285I   SYS69273.T115427.RV000.J063PGEX.GOSET         PASSED
IEF285I   VOL SER NOS= 231422.
IEF285I   SYS69273.T115427.SV000.J063PGEX.R0000016      SYSOUT
IEF285I   VOL SER NOS= M65294.
   END OF STEP 'GO      '   JOB 'J063PGEX'    STEPTIME=00.00.01  CLOCK=11.57.24   DATE=69.273   RETURN CODE=0000
IEF285I   SYS69273.T115427.RV000.J063PGEX.GOSET         DELETED
IEF285I   VOL SER NOS= 231422.
   END OF JOB  'J063PGEX'  I/O TIME=00.01.03   CPUTIME=00.00.11  CLOCK=11.57.24   DATE=69.273   SYSTEM PACKS 300 AND 311
```

(33)

Appendix B: Parameters of DD Statement

This appendix contains short descriptions of the parameters of the DD statement, arranged in alphabetical order of parameter names[1]. It describes the most frequently used facilities of each parameter, and summarizes their other features.

The format of the DD statement is described in Chapter 1, and the use of the ddname (the name of the DD statement) in Chapter 9. For a full description of the DD statement, refer to IBM System/360 Operating System: Job Control Language User's Guide, and Job Control Language Reference

Characteristics of the Data Set (DCB)

You can use the DCB parameter to add information about the characteristics of a data set to the data control block (DCB) that is constructed by one of the PL/I library subroutines when the associated file is opened. Code the keyword parameter

```
DCB=(list of characteristics)
```

Code the characteristics as keyword subparameters separated by commas, for example, DCB=(RECFM=FB,LRECL=80, ...).

To save time in coding the DCB parameter, you may be able to copy DCB information that already exists, either in the data set label of a similar data set or in an earlier DD statement in the job. To copy DCB information from the label of a cataloged direct-access data set, code the keyword parameter

```
DCB=dsname
```

Replace 'dsname' with the cataloged name of the data set. The volume that contains this data set must be mounted before execution of the job step containing the copy request.

If such a data set does not exist, you might still be able to copy the DCB parameter of an earlier DD statement in the job. To refer to this DD statement, code the keyword parameter

```
DCB=*.stepname.ddname
```

Replace 'stepname' and 'ddname' with the job step name and the DD statement name, respectively.

If the earlier DD statement is contained in a cataloged procedure step, you must include the procedure step name (i.e., DCB=*.stepname.procstepname.ddname).

If you want to modify the information that is copied from another data set label or DCB parameter, code

```
DCB=(reference,list of characteristics)
```

Replace 'reference' with 'dsname', or '*.stepname.ddname' or '*.stepname.procstepname.ddname'. The characteristics in the list override the corresponding copied attributes.

Subparameters of the DCB Parameter

The subparameters included in the DCB parameter correspond to the operands of the data management DCB macro instruction and are coded with the same keywords and values. A full description of the macro instruction appears in IBM System/360 Operating System: Supervisor and Data Management Macro Instructions; the following is a summary of those subparameters that can apply to a PL/I program. The notation used in the format descriptions is as follows:

'n' represents an unsigned decimal integer.

'|' indicates a choice of option.

Braces { } indicate that you must select one line from the items enclosed.

Items within brackets [ ] are optional; you may omit them at your discretion.

Code capital letters and numbers exactly as shown.

---

[1]For AFF, see SEP. For SPLIT and SUBALLOC, see SPACE.

BLKSIZE=n

specifies the length in bytes of a block. The maximum length is 32760 bytes.

For fixed-length records, the block size must be an integral multiple of the record length (LRECL); the minimum size is 1 byte.

For variable-length (V-format and VB-format) records, the block size must be at least eight bytes larger than the largest item of data that you expect to read or write (i.e., four bytes larger than the record length specified in LRECL). However, if the records are spanned (VS-format and VBS-format), you can specify block size independently of record length. The minimum block size for variable-length records is 18 bytes.

BUFNO=n

specifies the number of buffers to be used in accessing the data set. The maximum number is 255 (unless another maximum was determined for your installation during system generation). For a STREAM file or a BUFFERED RECORD file, if you do not specify the number of buffers or you specify zero buffers, the number is assumed to be two.

CODE=A|B|C|F|I|T|N

specifies the code in which paper tape was punched. (Data is read into main storage and then converted from that code to EBCDIC.)

A: ASCII (8-track)

B: Burroughs (7-track)

C: NCR (8-track)

F: Friden (8-track)

I: IBM BCD perforated-tape transmission code (8-track)

T: Teletype (5-track)

N: No conversion required (F-format records only)

If no code is specified, I is assumed.

CYLOFL=n

specifies, for an INDEXED data set, the number of tracks on each cylinder to be reserved for the records that overflow from other tracks in that cylinder. The theoretical maximum is 99, but the practical limit varies with the particular device.

There must be at least one track in each cylinder to hold the track index, and one to hold the prime data.

DEN=0|1|2|3

specifies the recording density for magnetic tape as follows:

DEN	Bytes per inch (bpi)	
	7-track	9-track
0	200	-
1	556	-
2	800	800
3	-	1600

The density assumed if you omit this subparameter is:

7-track: 800 bpi
9-track (single density): 800 bpi
9-track (dual density): 1600 bpi

(The subparameter TRTCH is required for 7-track tape.)

DSORG=IS|DA

specifies the organization of the data set you are creating:

IS (indexed sequential): INDEXED data set

DA (direct access): REGIONAL data set

This subparameter is not required for CONSECUTIVE data sets.

KEYLEN=n

specifies the length in bytes of the recorded key of records in INDEXED, REGIONAL(2), and REGIONAL(3) data sets. The maximum key length is 255 bytes.

LIMCT=n

limits the extent of the search for a record or space to add a record in a REGIONAL(2) or REGIONAL(3) data set beyond the region number specified in the source key.

If you do not specify a limit, the search starts at the specified region and continues through the whole of the data set.

For REGIONAL(2), LIMCT specifies the number of records to be searched. The search starts at the beginning of the track on which the record is situated and continues to the end of the track that contains the last record to be searched.

For REGIONAL(3), LIMCT specifies the number of tracks to be searched.

LRECL=n

specifies the length of a record in bytes; the maximum length is 32760 bytes for F-format records, and 32756 bytes for V-format records. You must specify a record length for blocked records.

For F-format and FB-format records, the record length must not exceed the block size (BLKSIZE) value; the minimum length is 1 byte.

For V-format records, give the maximum record length including the four control bytes required by the operating system; the minimum record length for V-format records is 14 bytes (ten bytes of data and four control bytes). The record length for V-format and VB-format records must be at least four bytes less than the block size (BLKSIZE) value; however, for VS-format and VBS-format records, it can be specified independently of block size. If the logical record length of any spanned variable-length record in a data set exceeds 32756, specify LRECL=X.

MODE=C|E

specifies the mode of operation for a card reader or punch: E indicates EBCDIC, and C specifies column binary. If you do not specify the mode, E is assumed.

NCP=n

specifies the number of channel programs allocated to a file when it is opened: the number of simultaneous input/output operations on the file (i.e., the number of incomplete event variables) cannot exceed the number of channel programs. The NCP subparameter applies only to DIRECT access to INDEXED data sets or UNBUFFERED SEQUENTIAL access to CONSECUTIVE or REGIONAL data sets. The maximum number of channel programs is 99 (unless another maximum was established for your installation at system generation); the default value assumed if you omit the subparameter is 1.

For DIRECT access to an INDEXED data set, simultaneous input/output operations in excess of the number of channel programs are queued until a channel program becomes available.

For UNBUFFERED SEQUENTIAL access to CONSECUTIVE or REGIONAL data sets, the ERROR condition is raised if there are too many simultaneous operations.

The NCP subparameter overrides the BUFNO subparameter or the BUFFERS option of the ENVIRONMENT attribute. One buffer is allocated for each channel program.

NTM=n

specifies, for an INDEXED data set, the number of tracks in the cylinder index referred to by each master index entry, and the number of tracks within each level of the master index referred to by each entry in the next higher level. The maximum value for n is 99.

OPTCD=option list

lists optional data management services. To indicate the services you require, code the appropriate letters (see below), without separating blanks, in place of 'option list' (e.g., OPTCD=LY).

OPTCD=C requests chained scheduling, which improves input/output performance by reducing the time required to transmit blocks to and from auxiliary storage devices. In chained scheduling, the data management routines bypass the normal input/output scheduling routines and chain several input/output operations together; a series of read operations, for example, is issued as a single chain of commands instead of several separate commands.

Chained scheduling is most useful in programs whose performance is input/output limited. If you use this feature, you should request at least three data management buffers or at least three channel programs. Chained scheduling can be used with CONSECUTIVE or REGIONAL SEQUENTIAL data sets; it should not be used for INPUT or UPDATE with U-format records.

OPTCD=I requests an independent overflow area for an INDEXED data set; you must define this overflow area in a separate DD statement.

OPTCD=L requests that a record in an

INDEXED data set be recognized as deleted if its first byte contains (8) '1'B.

OPTCD=M requests the creation of a master index in accordance with the information given in the NTM subparameter.

OPTCD=U suppresses the raising of the TRANSMIT condition when an invalid character is passed to a printer with the universal character set feature. A blank is printed in place of the invalid character.

OPTCD=W requests a write validity check for a direct-access device.

OPTCD=Y requests that the data management routines use the cylinder overflow area for overflow records in an INDEXED data set. The size of the overflow area is established by CYLOFL=n.

PRTSP=0|1|2|3

specifies the spacing required after each printed line; the default value is 1. (For example, PRTSP=3 causes two blank lines to appear between each printed line.) This subparameter is ignored if the record format includes ANS or IBM System/360 control characters.

RECFM= { F[B] [S] / V[B] [S] / U } [T] [A|M]

indicates the record format as follows:

F: Fixed-length records

V: Variable-length records

U: Undefined-length records

If you do not specify a record format, U-format is assumed, except for PRINT files, for which V-format is the default assumption.

The optional subfields are:

B: Blocked records.

S: Standard (fixed-length records only). No blocks, except possibly the last, will be shorter than the specified block size.

S: Spanned (variable-length records only). If variable-length records are spanned, the record length specified by LRECL can exceed the block size specified by BLKSIZE; if necessary, the records are segmented and the segments are placed in consecutive blocks. If the records are unblocked, each block contains only one record or segment; if the records are blocked, each block contains as many records or segments as it can accommodate.

T: Track overflow. Track overflow is an operating system feature that can be incorporated during system generation. It allows a block to overflow from one track of a direct-access device to another. Track overflow is useful in achieving greater data-packing efficiency, and also allows the size of a record to exceed the capacity of a track.

Note: You cannot use track overflow for REGIONAL(3) data sets with U-format or V-format records or for INDEXED data sets.

A: The first byte of each record contains an ANS printer/punch control character.

M: The first byte of each record contains an IBM System/360 printer/punch control character.

RKP=n

specifies, for an INDEXED data set, the position (n) of the first byte of an embedded key relative to the beginning of the record (byte 0). RKP=0 implies that the key is not embedded. (For example, if 'XYZ' is the key embedded in the record 'ABCXYZDEF',RKP=3.)

STACK=1|2

refers to a card reader or punch:

1. All cards read or punched are to be fed into stacker 1.

2. All cards read or punched are to be fed into stacker 2.

Stacker 1 is assumed if you omit this subparameter. If you want stacker 3, specify the ANS machine-code character in the RECFM parameter of the DD statement, and insert the appropriate character as the first data byte.

TRTCH=C|T|E|ET

is required when a data set is recorded or is to be recorded on a 7-track tape.

It specifies the recording technique to be used:

C: Data conversion, odd parity, no translation.

T: Translation, odd parity, no data conversion.

E: Even parity, no data conversion, no translation.

ET: Translation, even parity, no data conversion.

If you omit this subparameter, odd parity, no data conversion and no translation are assumed.

Notes:

1. Data conversion and translation: Data on 9-track magnetic tape, like that in main storage, is held in 8-bit bytes, a ninth bit being used for parity checking; data on 7-track tape is held in the form of 6-bit characters with a parity bit. The conversion feature of the 2400 series magnetic-tape drives treats all data as if it were in the form of a bit string, breaking the string into groups of six bits for writing on 7-track tape, or into groups of eight bits for reading into main storage. The translation feature changes the form in which character data is held from 8-bit EBCDIC to 6-bit BCD or vice versa. If you specify neither conversion nor translation, only the last six bits of each 8-bit byte are transmitted; the first two are lost on output and are set to zero on input.

2. Parity: Odd parity checking is normally employed in IBM System/360, but you should specify even parity if you want to read a tape that was written by a system using even parity, or to write a tape for a system that demands even parity.

3. Choice of technique: The use of a technique other than C restricts the character set in which data can be written if it is subsequently to be reread and result in the same bit configuration in main storage. (An 8-bit code offers 256 possible configurations, but a 6-bit code only 64.) For stream-oriented or record-oriented transmission of character strings or pictured data, you can use technique C or T; you can also specify ET if your program is written in the 48-character set. (Seven-track tape recording systems indicate a zero bit by the absence of magnetization of the tape. Even parity checking does not allow the code 000000 to be used to represent the character zero, since an unmagnetized band is not acceptable on the tape. Therefore the code that would otherwise represent a colon (:) is used for the character zero, precluding the use of the full PL/I 60-character set.) For record-oriented transmission of arithmetic data, you must specify technique C.

Postponing Definition of Data Set (DDNAME)

A DD statement in a cataloged procedure need not contain descriptive parameters. Instead, it can point to a subsequent DD statement that contains a complete description of the data set. To postpone the definition of a data set, code:

```
DDNAME=ddname
```

Replace 'ddname' with the name of the DD statement that will contain the complete information.

Data Set Status and Disposition (DISP)

The DISP parameter (which can have three subparameters) describes the status of a data set and indicates what is to be done with it after termination of the job step that processes it. You can omit this parameter if a data set is created and deleted during a single job step.

The first subparameter shows the status of the data set with respect to the job step. If the data set is created in the job step, code

```
DISP=NEW
```

(or code DISP=MOD and include parameters usually required by new data sets (see below) or omit the status specification altogether).

To specify an existing data set to be used as input to a program, code:

```
DISP=OLD
```

If the data set resides on a direct-access volume and is part of a job whose operations do not preclude simultaneous use of the data set by another job, code

```
DISP=SHR
```

which has meaning only in a multiprogramming environment. Once a data set has been given the status SHR, every reference to the data set within the job must specify the same status or the data set will be considered unusable by concurrent jobs. If SHR is coded in other than a multiprogramming environment, the system assumes the status of the data set to be OLD.

If you want to extend a CONSECUTIVE data set by adding records at the end, code:

```
DISP=MOD
```

When the data set is opened, the read/write mechanism is automatically positioned after the last record in the data set. If the operating system is unable to find the data set, it assumes that it does not yet exist: the first time you open the data set, the operating system assumes DISP=NEW, and thereafter DISP=MOD.

The second subparameter indicates how you want the data set handled by the job scheduler after normal termination of the job step. (If you omit the first, indicate its absence by placing a comma in front of the second.) If you want the data set to assume the status that it had before the job step, you need not code the second: data sets that existed before the job step began will continue to exist, and data sets created in the job step, or earlier in the job, with DISP=(,PASS) will be deleted. If there is any uncertainty, code both subparameters in full.

To catalog a data set, code:

```
DISP=(status,CATLG)
```

'Status' shows the status of the data set as discussed above. When you request cataloging, an index entry pointing to the data set is placed in the system catalog. DD statements in subsequent jobs can then refer to this data set simply by giving its fully qualified name and the disposition.

To uncatalog an input data set, code:

```
DISP=(OLD,UNCATLG)
```

The catalog entry that points to the data set is removed from the index. If the data set resides on a direct-access volume, it remains tabulated in the volume table of contents (VTOC).

If you have no further need for a data set after its use, code:

```
DISP=(status,DELETE)
```

The data set is automatically uncataloged if you have used the catalog to locate it. In addition, the system removes the VTOC entry associated with the data set, if it resides on a direct-access device. If you code DISP=(SHR,DELETE), the system assumes OLD instead of SHR.

For data sets that are used in a later job but are not of sufficient importance to warrant their being cataloged, code:

```
DISP=(status,KEEP)
```

The data set is kept intact until the system encounters DISP=(status,DELETE). If the volume containing the data set is demounted, the system advises the operator of the KEEP disposition. If the data set resides on a direct-access volume, it remains tabulated in the volume table of contents.

When a data set is used by two or more job steps in the same job, you can eliminate retrieval and disposal operations by passing it from step to step. You do not indicate the final disposition of the data set until its last use in the job. To pass a data set to a succeeding step, code:

```
DISP=(status,PASS)
```

Subsequent DD statements referring to the passed data set must identify it with the DSNAME parameter, must provide either no unit information, or unit information consistent with that in the original data set, and must issue another disposition. Between steps, the volume that contains the passed data set remains mounted. However,

if the system or an intervening step requires that device, the volume may be dismounted and saved by the operator; it is remounted when required in a succeeding job step. To ensure that the volume remains mounted during steps in which it is not required, use dummy DD statements referring to the data set in these steps.

The third subparameter indicates how you want the data set handled by the job scheduler at the end of the job step if the job step terminates abnormally. If you omit this subparameter, the disposition requested in the second subparameter will be performed if the job step terminates abnormally.

If in the event of abnormal termination you wish to have a data set cataloged, code:

```
DISP=(status,disposition,CATLG)
```

'Status' and 'disposition' show the status and disposition of the data set at normal end of job, as discussed above.

If in the event of abnormal termination you wish to uncatalog a data set, code:

```
DISP=(OLD,disposition,UNCATLG)
```

'Disposition' refers to the disposition you want made of the data set upon normal termination of the job step.

If in the event of abnormal termination you wish a data set to be deleted, code:

```
DISP=(status,disposition,DELETE)
```

If in the event of abnormal termination you wish a data set to be kept, code:

```
DISP=(status,disposition,KEEP)
```

The second subparameter can be omitted. If you code

```
DISP=(OLD,,DELETE)
```

the default for the second subparameter follows the rule established above.

Note: The third (conditional disposition) subparameter applies only if the job step is terminated abnormally. It does not apply if a 'STEP WAS NOT EXECUTED' message is given, either due to a JCL error or a condition code setting.

Identifying the Data Set (DSNAME)

The DSNAME parameter identifies the data set to which the DD statement refers. You need not code this parameter if the data set is temporary or resides on a unit-record device or on an unlabeled magnetic tape; the system automatically assigns a temporary name. You can specify the DSNAME parameter in one of three ways:

1. To name or retrieve a data set that will be identified in later jobs by name, or that was assigned a name in an earlier job or job step, code:

   ```
   DSNAME=dsname
   ```

   The word DSNAME can be abbreviated to DSN. Replace 'dsname' with the cataloged or tabulated name of the data set. If the catalog has more than one level of index, you must give a fully qualified name (e.g., A.B.LINKFILE).

   If the DD statement refers to a particular generation of a generation data group, you must code the generation number in parentheses, i.e., DSNAME=dsname(number).

   If the DD statement refers to a member of a partitioned data set, you must code the member name in parentheses after dsname, i.e., DSNAME=dsname(membername).

   If the DD statement is one of a group of DD statements that define an INDEXED data set, you must code one of the terms INDEX, PRIME, or OVFLOW in parentheses after dsname (e.g., DSNAME=dsname(PRIME)).

2. To obtain the data set name from an earlier DD statement, code:

   ```
   DSNAME=*.stepname.ddname
   ```

   Replace 'stepname' and 'ddname' with the job step name and DD statement name, respectively, where the data set was first defined.

   If the earlier DD statement is contained in a cataloged procedure

step, you must include the procedure step name, i.e., DSNAME=*.stepname.procstepname.ddname.

3. A data set that exists only within the boundaries of a job can be assigned any temporary name. To assign a temporary name, code:

```
DSNAME=&&name
```

Replace 'name' with any name of not more than eight characters not used by another temporary data set in the job (e.g., DSNAME=&&TEMPDSET). The system replaces the &&name with a name of the form 'jobname.dsname'.

If the DD statement refers to a member of a temporary partitioned data set, you must code the member name in parentheses, i.e., DSNAME=&&name(membername).

If the DD statement is one of a group of DD statements that define a temporary INDEXED data set, you must code one of the terms INDEX, PRIME, or OVFLOW in parentheses after the &&name (e.g., DSNAME=&&name(PRIME)).

Data Set Label (LABEL)

Magnetic-tape volumes can contain standard or nonstandard volume labels and data set header and trailer labels, or they may have no labels at all. Direct-access devices have standard labels. The LABEL parameter indicates the position of a data set relative to the other data sets on a tape reel, and the type of labels; it can also specify the retention period for a data set on magnetic tape or direct-access device, and whether a password is required before it can be accessed. Only the sequence number and label type are described here.

For a magnetic-tape volume, if a data set is not first in sequence on the reel, the LABEL parameter must include a data set sequence number to position the tape properly. The sequence number describes the position of the data set relative to other data sets on the volume or group of volumes. Code:

```
LABEL=seq#
```

Replace 'seq#' with the sequence number (1-4 decimal digits) of the data set on the reel.

To create or retrieve a data set that does not have standard labels, you must include the LABEL parameter. To specify the type of labels, code:

```
LABEL=(,type)
```

Replace 'type' with:

SL - if the data set has standard labels

NL - if the data set has no labels

NSL - if the data set has nonstandard labels

SUL - if the data set has both standard and user labels

BLP - to bypass label processing

If you specify SUL, SL, or omit the label type (in which case standard labels are assumed), the operating system will ensure that the correct volumes are mounted. If you specify NSL, your installation must have incorporated label processing routines into the operating system. If you specify NL, the data set must have no labels. (Specifying NL for an output data set on a labeled volume can cause the volume label to be erased.)

The feature that allows you to bypass label processing is a system generation option (OPTIONS=BYLABEL). If this option was not requested at system generation and you have coded BLP, the system assumed NL.

Note that the two LABEL subparameters are positional. (If you omit the first, indicate its absence by placing a comma in front of the second.)

Optimizing Channel Usage (SEP and AFF)

A job step that requires several input and output operations might be performed more efficiently by balancing the channel requirements of its data sets. To obtain optimum channel usage, you can request that a data set be assigned a separate channel from the ones assigned to earlier data sets. A later DD statement can express the same separation requirements by requesting affinity.

To request channel separation from as many as eight other data sets in the job step, code:

```
SEP=(ddname,...,ddname)
```

Replace 'ddname' with the names of up to eight earlier DD statements in the job step.

To obviate writing identical SEP parameters for different data sets, you can request affinity with an earlier data set that requested channel separation by coding

```
AFF=ddname
```

in a later DD statement. Replace 'ddname' with the name of the earlier DD statement. The data set that requests affinity is also allocated a separate channel from those identified in the SEP parameter of the earlier statement (but not necessarily the same channel as the data set to which the SEP parameter referred).

The operating system will comply with your requests for separation and affinity only if enough channels are available.

Note: Do not confuse the SEP and AFF parameters with the subparameters SEP and AFF of the UNIT parameter.

Allocating Direct-Access Space (SPACE, SPLIT, and SUBALLOC)

When creating a new data set on a direct-access volume, you must indicate in your DD statement how much space the data set will need. You can allocate space:

1. By requesting the amount of space and letting the system assign specific tracks.
2. By requesting specific tracks.
3. By splitting cylinders with other data sets.
4. By suballocating space from an earlier data set.

The most frequently used technique of space allocation allows you to specify the amount of space you need and let the system assign specific tracks. Options permit you to specify the manner in which the space is to be arranged, to release unused space, and to request that the space begin and end on cylinder boundaries.

You can specify the amount of space in units of tracks, cylinders, or blocks, whichever is most convenient. In the last case the system will compute the number of tracks or cylinders required. To allocate space using this technique, code:

```
SPACE=(units,quantity)
```

Replace 'units' with:

TRK - if you want space in tracks

CYL - if you want space in cylinders

Average block length in bytes - if you want space in terms of blocks

Replace 'quantity' with the amount of space you need in the units you have chosen (e.g., SPACE=(TRK,200) for 200 tracks, SPACE=(CYL,10) for ten cylinders, and SPACE=(400,100) for 100 blocks with an average length of 400 bytes).

Notes:

1. For most efficient performance, request space in units of cylinders (CYL), or in units of blocks with the ROUND subparameter.
2. The average block length cannot exceed 65,535 bytes.
3. If you request space in units of blocks, and the blocks have keys, you should specify an average block requirement which includes the key.
4. Space allocation by blocks allows device independence.

If the possibility exists that the data set might at some time exceed the amount of space you requested, you can ensure that extra space will be made available by denoting an incremental quantity. Code:

```
SPACE=(units,(quantity,increment))
```

Replace 'increment' with a decimal number. Each time the data set exhausts its space, additional space will be allocated on the same volume in the amount of the increment. For example, if you code SPACE=(TRK,(200,10)), the data set would be initially allocated 200 tracks. If it later exceeded 200 tracks, ten additional tracks would be made available. This incrementing by ten tracks would take place each time the data set exhausted its total space (up to a maximum of 15 times).

If the data set for which you are allocating space has partitioned

organization, you must indicate the size of its directory in the SPACE parameter. Code:

```
SPACE=(units,(quantity,,directory))
```

Replace 'directory' with the number of 256-byte blocks in the directory. If you wish to give an incremental quantity, code SPACE=(units,(quantity,increment, directory)).

If the data set has INDEXED organization, you can indicate the size of its index in the SPACE parameter. Code:

```
SPACE=(units,(quantity,,index))
```

Replace 'index' with the size of the index, in cylinders. If you do not allocate space for the index , the operating system will use part of the independent overflow area or prime data area for the cylinder and master indexes.

You cannot give an incremental quantity for an INDEXED data set, but you can do so for a REGIONAL data set only for SEQUENTIAL creation.

you can release unused space when you have finished writing a data set by including the positional subparameter RLSE in the SPACE parameter. Code

```
SPACE=(units,(quantities),RLSE)
```

where 'quantities' represents 'quantity, increment,directory' or 'quantity, increment,index' as used above. RLSE cannot be used with ISAM data sets.

The SPACE parameter also allows you to request that the data set be placed on contiguous tracks or cylinders, and to place a data set in a specific position on the volume.

When a job step involves one or more data sets that have corresponding records, you can minimize access-arm movement by using the SPLIT parameter instead of the SPACE parameter. The SPLIT parameter requests that each data set be given a proportion of the tracks on every cylinder allocated, allowing access to corresponding records in the data sets without movement of the access arm.

Another method of obtaining direct-access space is through the technique of suballocation, using the SUBALLOC parameter. Suballocation allows you to place a number of data sets in contiguous order on a direct-access device.

Routing Data Sets through an Output Stream (SYSOUT)

The operating system provides output streams through which you can route data sets destined for unit record devices. To route a data set through any output stream, code:

```
SYSOUT=x
```

Replace 'x' with an alphabetic (A-Z) or numeric (0-9) character. The character you select specifies an output class and a corresponding output stream. (The computer operator allocates the output classes; a usual convention is that class A refers to a printer and class B to a card punch.)

When using an operating system with PCP, to route a data set through the output stream that carries system messages, code:

```
SYSOUT=A
```

You can also route both data sets and system messages through the same output stream when using a system with MFT or MVT. To do this, you must specify in the SYSOUT parameter the output class that you selected for the MSGCLASS parameter in your JOB statement.

Operating systems with MFT and MVT have two additonal options for the SYSOUT parameter. They allow you to name your own special program to handle ouput operations, and to select a specific type of output form for a printed or punched data set.

Specifying a Character Set (UCS)

The Universal Character Set (UCS) parameter allows you to have your output printed with a specified character set on a printer with the UCS feature. This character set is selected from a library of the character sets available at your installation; such a library would only exist if your installation had a UCS printer. Each character set is held in the library as an image of the appropriate subset of the universal character set of 240 graphic symbols. The selected image is loaded into

the UCS buffer of the control unit for the printer, which must be equipped with the appropriate print chain or train.

To specify a required character set code:

```
┌──────────────────────────────────────┐
│        UCS=character set code        │
└──────────────────────────────────────┘
```

Replace 'character set code' with one of the twelve identification codes for the IBM standard character sets or with a code for your own character set. These codes must be one to four bytes long; those for the PL/I character sets are:

PN Alphameric (PL/I)

QNC Alphameric (PL/I commercial)

QN Alphameric (PL/I scientific)

If you omit the UCS parameter when using a printer with the UCS feature, there may be a default character set for your installation. If not, a message will be put on the console asking the operator to specify a character set.

Other UCS features available allow you to transpose certain EBDIC characters with others, and to request the operator to verify that the character set chosen is the right one.

Requesting a Unit (UNIT)

The UNIT parameter of the DD statement allows you to specify information about the input or output unit(s) used by a data set. You can identify a specific unit or group of units by its address, its type number, or its group name.

To identify a unit by its address, code:

```
┌──────────────────────────────────────┐
│            UNIT=address              │
└──────────────────────────────────────┘
```

Replace 'address' with the 3-byte address of the unit (e.g., UNIT=180 for channel 1, control unit 8, unit 0).

To identify a unit by its type number, code:

```
┌──────────────────────────────────────┐
│             UNIT=type                │
└──────────────────────────────────────┘
```

Replace 'type' with a valid unit type number (e.g., UNIT 2400-2). The following are valid unit type numbers:

Tape Units

2400	2400 series 9-track magnetic-tape drive that can be allocated to a data set written or to be written with a density of 800 bpi.
2400-1	2400 series magnetic-tape drive with 7-track compatibility and without data conversion.
2400-2	2400 series magnetic-tape drive with 7-track compatibility and data conversion.
2400-3	2400 series 9-track magnetic-tape drive that can be allocated to a data set written or to be written with a density of 1600 bpi.
2400-4	2400 series 9-track magnetic-tape drive having an 800 and 1600 bpi (density) capability.

Direct-Access Units

2301	2301 drum storage unit
2302	2302 disk storage drive
2303	2303 drum storage unit
2311	2311 disk storage drive
2314	2314 direct-access storage facility
2321	Bin mounted on 2321 data cell drive

Unit Record Equipment

1052	1052 printer keyboard
1403	1403 printer or 1404 printer (continuous form only)
1442	1442 card read punch
1443	1443 printer
2501	2501 card reader
2520	2520 card read punch
2540	2540 card read punch (read feed)
2540-2	2540 card read punch (punch feed)
2671	2671 paper-tape reader

Graphic Units

1053	1053 model 4 printer
2250-1	2250 display unit, model 1
2250-3	2250 display unit, model 3
2260-1	2260 model 1 display station (local attachment)
2260-2	2260 model 2 display station (local attachment)
2280	2280 film recorder
2282	2282 film recorder/scanner

At system generation, your installation can designate names for individual units or collections of units (for example, to classify collections of magnetic-tape and direct-access units under the same name). To identify such a group of units in a DD statement, code:

```
UNIT=group
```

Replace 'group' with a valid unit group name (e.g., UNIT=TAPE). If your installation uses IBM-supplied cataloged procedures, the following group names must have been established at system generation:

Name	Types of Devices
SYSSQ	Magnetic tape, direct access
SYSDA	Direct access
SYSCP	Card punch

Other subparameters of the UNIT parameter allow you to specify the number of units you need, defer mounting of volumes until the data set is opened, and request that the data set not be retrieved or stored by access mechanisms used by certain other data sets.

Specifying Volume Information (VOLUME)

To request specific volumes, you can either identify the volumes by their serial numbers or use the volumes used by an earlier data set in the job.

To identify volumes by their serial numbers, code:

```
VOLUME=SER=(ser#,...,ser#)
```

VOLUME can be abbreviated to VOL. Replace 'ser#' with the serial numbers associated with the volumes; a serial number may consist of up to six characters. You must use this form of the VOLUME parameter when retrieving noncataloged data sets. If only one volume is involved, you can omit the parentheses, i.e., code VOLUME=SER=ser#.

There are two ways of requesting the volumes used by an earlier data set. If the data set is cataloged or passed, and you wish to refer to it, code:

```
VOLUME=REF=dsname
```

Replace 'dsname' with the name of the data set. If the data set is not cataloged or passed, and you wish to refer to the DD statement that defines it, code:

```
VOLUME=REF=*.stepname.ddname
```

Replace 'stepname' and 'ddname' with the name of the job step and DD statement where the earlier data set is defined. If the earlier data set is part of a cataloged procedure, you must include the procedure step name, i.e., VOLUME=REF=*.stepname.procstepname.ddname.

Other subparameters of the VOLUME parameter allow you to request private volumes, request that private volumes remain mounted until the end of the job, select volumes when the data set resides on more than one, and request more than one nonspecific volume.

Appendix C: Versions of the PL/I (F) Compiler

This edition, Form C28-6594-6 of the PL/I (F) Programmer's Guide, documents the fifth version of the compiler with the improvements incorporated for Release 19 of the operating system.

Earlier versions are:

1st version: Form C28-6594-0
2nd version: Form C28-6594-1
3rd version: Form C28-6594-2
4th version: Form C28-6594-3
4th version, release 17: Form C28-6594-4
5th version: Form C28-6594-5

The more important differences between these versions of the compiler are listed below. There then follows a statement concerning the compatibility between compiled code and library modules of various versions.

Changes at Second Version

The most significant changes for the second version of the compiler are:

RECORD I/O: The statements: READ, WRITE, REWRITE, and DELETE.

The attributes: RECORD, UPDATE, SEQUENTIAL, DIRECT, BACKWARDS, BUFFERED, UNBUFFERED, and KEYED.

The ON-conditions: RECORD and KEY.

The built-in functions: ONFILE and ONKEY.

Note: The usage UPDATE SEQUENTIAL was not supported except for INDEXED data set organization.

COMPILE-TIME PROCESSING: The compile-time processing feature of PL/I.

COMPILER OPTIONS: Abbreviated names as alternatives to the full names for compiler options.

ARRAY INITIALIZATION: Initialization of arrays of STATIC variables by means of the INITIAL attribute.

STREAM I/O: The options PAGESIZE and LINESIZE.

The ON-condition NAME.

LIST/DATA-DIRECTED OUTPUT: Alignment of data on preset tab positions.

RECORD FORMAT: The use of undefined-format source records.

PAPER TAPE: Paper tape as input to the compiler and object program.

OPERATORS: The operators ¬> and ¬<, and their 48-character set equivalents NG and NL.

QUALIFIED NAMES: The resolution of apparently ambiguous name qualification.

OBJECT PROGRAM LISTING: Double-column format for the object program listing.

OBJECT-TIME ERROR HANDLING: Optional inclusion of the statement number in object-time diagnostic messages.

Combination of SNAP output with SYSTEM action for ON statements.

RECURSION ENVIRONMENTS: A change in the interpretation of ENTRY parameters and ON units in recursive contexts.

CATALOGED PROCEDURES: A new cataloged procedure (PL1LFLG) for link-editing and execution.

Changes at Third Version

The most significant changes for the third version of the compiler are:

OBJECT PERFORMANCE: Changes in the object code generated by the compiler will result in considerable improvements in the object-time performance. The most significant improvements are in the following areas: data conversions, the SUBSTR function and pseudo-variable, the INDEX function, the UNSPEC function, object-time error handling and procedural housekeeping, and the use of GO TO label variables.

ARRAY INITIALIZATION: Initialization of arrays of AUTOMATIC or CONTROLLED

variables by means of the INITIAL attribute.

UPDATE SEQUENTIAL: The use of UPDATE SEQUENTIAL for CONSECUTIVE and REGIONAL data set organizations.

ASYNCHRONOUS OPERATION: The EVENT option on I/O statements, the COMPLETION built-in function and pseudo-variable, and the WAIT statement.

BATCHED COMPILATION: The facility for batched compilation of programs and a new compiler option, OBJNM.

LINK-EDITING: A changed method of link-editing library routines into an object program, facilitating both the link-editing of PL/I object modules from a library and the use of overlay technique with PL/I object modules.

MIXED DEFINING: The severity of diagnostic messages for defined data of type different from the type of the base is reduced from terminal to error, permitting the compilation of programs using mixed defining.

Changes at Fourth Version

The most significent changes for the fourth version of the compiler are:

LOCATE I/O AND LIST PROCESSING: The following language is now supported:

Statements and options:

```
READ FILE(filename)
 SET(pointer-variable)
 [KEY(expression)|KEYTO
 (character-string-variable)];

LOCATE based-variable
 FILE(filename)
 [SET(pointer-variable)]
 [KEYFROM(expression)];

REWRITE FILE(filename);

ALLOCATE based-variable
 [in(area-variable)]
 [SET(pointer-variable)];

FREE based-variable
 [IN(area-variable)];
```

Assignment:

```
AREA to AREA

POINTER/OFFSET to POINTER/OFFSET
```

Attributes:

```
AREA[(expression)]
BASED(pointer)
OFFSET(based-variable)
POINTER
REFER(identifier)
```

Built-in functions:

```
ADDR
EMPTY
NULL
NULLO
```

Condition:

```
AREA
```

Operation:

```
-> in 60-character set
PT in 48-character set
```

ASYNCHRONOUS OPERATIONS AND MULTITASKING: The following language is now supported:

Statements and options:

```
CALL statement with TASK, EVENT,
 and PRIORITY options in any
 combination

WAIT statement extended to allow
 array names in the event list

DISPLAY statement with REPLY and
 EVENT options

UNLOCK statement

NOLOCK option in READ statement
```

Assignment:

```
EVENT to EVENT
```

Attributes:

```
EVENT
EXCLUSIVE
TASK
```

Built-in functions/pseudo-variables:

```
COMPLETION
PRIORITY
STATUS
```

Multitasking is supported by the MVT system

Multiprocessing

DATA INTERCHANGE: The COBOL option in the ENVIRONMENT attribute; the ALIGNED/UNALIGNED attributes (for FORTRAN data interchange).

ASSEMBLER SUBROUTINES: A variable-length argument list can be passed to assembler subroutines invoked by a PL/I program.

STRING HANDLING: The STRINGRANGE condition for use with SUBSTR; the STRING function.

STREAM I/O: LINESIZE, SKIP, and COLUMN in non-PRINT files; PUT DATA with no data list.

RECORD I/O: Some types of VARYING string may be used with the INTO or FROM options; the KEY option in the DELETE statement is now optional. The DELETE statement is now supported for INDEXED data sets using SEQUENTIAL access. The new ENVIRONMENT options (INDEXAREA, NOWRITE, and REWIND) provide improved performance.

COMPILER OPTIONS: SIZE and SORMGIN have been changed, and four new options (OPLIST, EXTDIC, MACDCK, and NEST) have been added.

COMPILE-TIME OPTIMIZATION: Macro-processor concatenations are improved.

OBJECT PROGRAM OPTIMIZATION: Constant subscript and constant expression evaluation; some instances of VARYING strings in assignment; in-line code for some VARYING string operations; prologue optimization; in-line handling of certain data conversions and some bit-string assignments; rounding-off (instead of truncation) for E- and F-format output; dope vector initialization improved; optimization of some IF statements

LISTING IMPROVEMENTS: More details in attribute listings; aggregate listing in alphabetical order; sizes of the STATIC and program control sections are given; the size of each DSA is given; statement number provided in diagnostic message for invalid pictures; improvements in aggregate length table for BASED items.

PROGRAM RESTART: The operating system checkpoint/restart facility is available under PCP.

EVALUATION OF EXPRESSIONS: The order of priority is changed; concatenation now comes before the comparison and logical operators in the sequence of priority.

PL/I SORT: The operating system sort program is available for use with PL/I programs.

Note: In multitasking the action taken by the conditions ERROR and FINISH, and the statements EXIT and STOP vary significantly between the third and fourth versions. The following descriptions refer to the fourth version.

ERROR condition: if raised in a major task, FINISH is raised; if in any other task, that task is terminated.

FINISH condition: execution of the interrupted statement is resumed.

EXIT statement: causes immediate termination of the task containing this statement, and all tasks attached by this task. If in a major task, it has the same effect as STOP.

STOP statement: raises FINISH and causes immediate termination of the major task and all subtasks.

Changes at Fourth Version, Release 17

The most significant changes for the fourth version at Release 17 are:

RECORD I/O: Spanned records (VS- or VBS-format) can be specified to span blocks.

Generic keys (GENKEY option) can be specified to access groups of records on an INDEXED data set.

PL/I SORT: User control or SORT ddnames for multiple use of PL/I SORT within a single job step is provided.

MULTIPROCESSING: More than one PL/I task may be executed simultaneously by a multiprocessing system.

PROGRAM RESTART: Improved checkpoint/restart facilities are supported by PCP and MVT systems.

CATALOGED PROCEDURES: Changes to some condition codes and to the dsnames for temporary data sets have been

incorporated into the PL/I cataloged procedures.

Changes at Fourth Version, Release 18

The most significant changes for the fourth version at Release 18 are:

COMPILER STATISTICS: Additional information is given in the listing to facilitate the automatic collection of statistics about the use of the operating system.

DEDICATED WORKFILES: With the MVT control program, a dedicated workfile can be substituted for workspace data sets.

LINKAGE LOADER: A new operating system service program, the linkage loader, can be used as an alternative to the linkage editor.

CATALOGED PROCEDURES: Two new cataloged procedures are introduced: PL1LFCG and PL1LFG; both for use with the linkage loader.

MULTIPLE CONSOLE SUPPORT (MCS): For an installation that has MCS, system messages can be displayed on one or more consoles.

MESSAGE LEVEL PARAMETER: The MSGLEVEL parameter in the JOB statement can have additional values.

Changes at Fifth Version

The most significant changes for the fifth version are:

TELEPROCESSING: The following language is now supported:

File attributes:

ENVIRONMENT Options: G(max. length) R(max. length)

TRANSIENT

ON-conditions:

PENDING

ASYNCHRONOUS OPERATIONS AND MULTITASKING: The following is now supported:

Task management:

Protection between tasks when executing 'soft' code in a multiprocessing environment

PRTY parameter in the EXEC statement

PRIORITY pseudo-variable enables priority of any task to be changed

OBJECT PROGRAM OPTIMIZATION: Optimization of DO-loops for faster operations; implementation of halfword binary; some in-line conversions; improved assignment code for fixed decimal data; improved edit I/O for F- and E-formats; improved register allocation.

OBJECT PROGRAM TERMINATION: The specified task asynchronous exit (STAE) feature for abnormal terminations in single and multitasking.

DATA SET DESCRIPTION: TRKOFL (track overflow) and NCP (number of channel programs) have been added to the list of options in the ENVIRONMENT attribute.

PROGRAM RESTART: The operating system checkpoint/restart facility is now available under PCP, MFT, and MVT.

IMPROVED FLOATING-POINT ENGINEERING CHANGE (IFPEC): Implementation of this feature has produced increased precision and faster execution for computational subroutines.

DATA EDITING:

Built-in functions:

TRANSLATE
VERIFY

PL/I LIBRARY: Selected library modules may now be stored permanently in main storage for MFT and MVT.

STRING HANDLING: STRING pseudo-variable.

FUNCTION REFERENCES: Mandatory RETURNS attribute on PROCEDURE and ENTRY statements.

BASED STRUCTURES: Improved precision for FIXED BINARY data used with the REFER option.

CATALOGED PROCEDURES: New cataloged procedures (PL1LFG and PL1LFCG); dedicated workfiles.

ABBREVIATIONS: The following abbreviations for file attribute keywords are accepted:

Keyword	Abbreviations
BUFFERED	BUF
EXCLUSIVE	EXCL
SEQUENTIAL	SEQL
UNBUFFERED	UNBUF

Changes at Fifth Version, Release 19

The most significant changes for the fifth version at Release 19 are:

In-stream procedure: Provision to test programmer-written procedures as in-stream procedures before being placed in the procedure library.

Model 195 support: New compiler options to support Model 195.

Step Abend facility: Enables system to issue a STEP ABEND after an abnormal termination from any task when the termination is caused by the ERROR condition, and no ERROR on-unit is established.

Syntax Check option: Provision made to terminate compilation after the syntax checking is completed, conditional on the error severity specified.

7 track tape default: Default changed from 200 b.p.i., to 800 b.p.i.

Changes at Fifth Version, Release 20

All changes at the fifth version for Release 20 are minor changes resulting from maintenance of the compiler and this publication.

Compatibility between Different Versions of the PL/I Library and Compiled Code

Certain changes and improvements have been made to PL/I, the compiler, and the library between the five versions of the compiler. As a result, certain incompatibilities have unavoidably arisen between library modules of the different versions. The purpose of this compatibility statement is to make clear to the user what incompatibilities exist, and how the problems raised by them can be overcome.

Several changes in the fifth version prevent this version from being completely compatible with earlier versions:

1. Fixed binary variables with a precision less than 16 are now aligned on halfword boundaries and are two bytes long. Fixed binary variables with a precision of 16 or more are aligned, as before, on fullword boundaries and are four bytes long. Therefore, the boundary alignments obtained in this version of the compiler for fixed binary data with a precision of less than 16 may differ from those obtained in a previous version. Programs based on previous versions may have to be recompiled if they contain fixed binary data, or make calls to modules IHESRT, IHEDUM, IHESAR, and IHETSA. These modules should have their fixed binary arguments re-declared with a precision greater than 15.

2. A change to the multitasking support in this version will provide faster execution for the PL/I user in a multiprocessing environment. Re-compilation is not necessary; re-link-edit is.

3. If a function reference returns a value, the entry-point name for the function reference must be declared with the RETURNS attribute. Programs compiled by earlier versions of the compiler without such a declaration will generate a warning message.

4. The restrictions on fixed binary variables in the REFER option have now been lifted, and such variables may be fixed binary integer variables of any precision. Both variables must be of the same precision.

Two definite compatibility statements can be made about the compilers in general:

1. Compiled code from any version of the compiler must always be executed using a library of the same version or a later version.

2. Library modules of different versions can be mixed only in the following circumstances:

   a. All link-edited modules must be of the same version as each other, and

   b. All dynamically linked or loaded modules must be of the same version as each other and must be of at least as late a version as the link-edited modules.

Unless a user has link-edited PL/I external procedures with modules from a PL/I library and placed them in a private

library for future use with main programs compiled by a later version of the compiler, these incompatibilities should cause no problems. Provided the user has installed the latest compiler and library components, all future link-editing operations will result in the incorporation of the correct library modules. If, however, he has link-edited some of his external procedures, then, if he intends to use them in conjunction with a main program containing later- version library modules, he must remove the earlier-version library modules from them.

Two methods may be employed to carry this out, one temporary and the other permanent:

1. The linkage editor map for the external procedure is examined to see whether any library modules have been incorporated in the load module; these can be identified by the initial letters IHE. If there are no library modules present in the load module, no further action is required. If library modules are present, then every time a main program needs to use the external procedure, it should be link-edited with it, using INCLUDE cards naming the library modules which are to be replaced (i.e., all of them) and an INCLUDE card naming the external procedure itself. (The latter must be a separate card and it must follow the INCLUDE card for the library modules.) This will result in the incorporation of the correct later-version library modules and then the external procedure itself. This method is temporary.

2. The permanent method is to link-edit the external procedure using the replace facility on the NAME card. In other words, the linkage editor is executed using the INCLUDE cards naming the library modules which are to be replaced, followed by an INCLUDE card naming the external procedure, followed by a NAME card naming the external procedure with the replace option.

Appendix D: System Requirements

Control Program Options

Figure D-1 shows the control program options which may be added in order to provide greater performance and/or programming flexibility for the various features of PL/I. These exclude input/output features.

PL/I Feature	Control Program Option[1]
WAIT	Multiple WAIT
Tasking	MVT
TIME, DATE	Timing: A. Time
TIME, DATE, DELAY	Timing: B. Interval timing

[1]See the publication IBM System/360 Operating System: Storage Estimates for further details of these options.

Figure D-1. Control Program Options

Machine Requirements

The PL/I (F) compiler requires an IBM System/360 Model 30 or higher with at least 64K bytes of main storage, of which at least 44K bytes should be allocated to the compiler by means of the SIZE compiler option. The PL/I (F) compiler also requires at least one direct-access device, and an operator console.

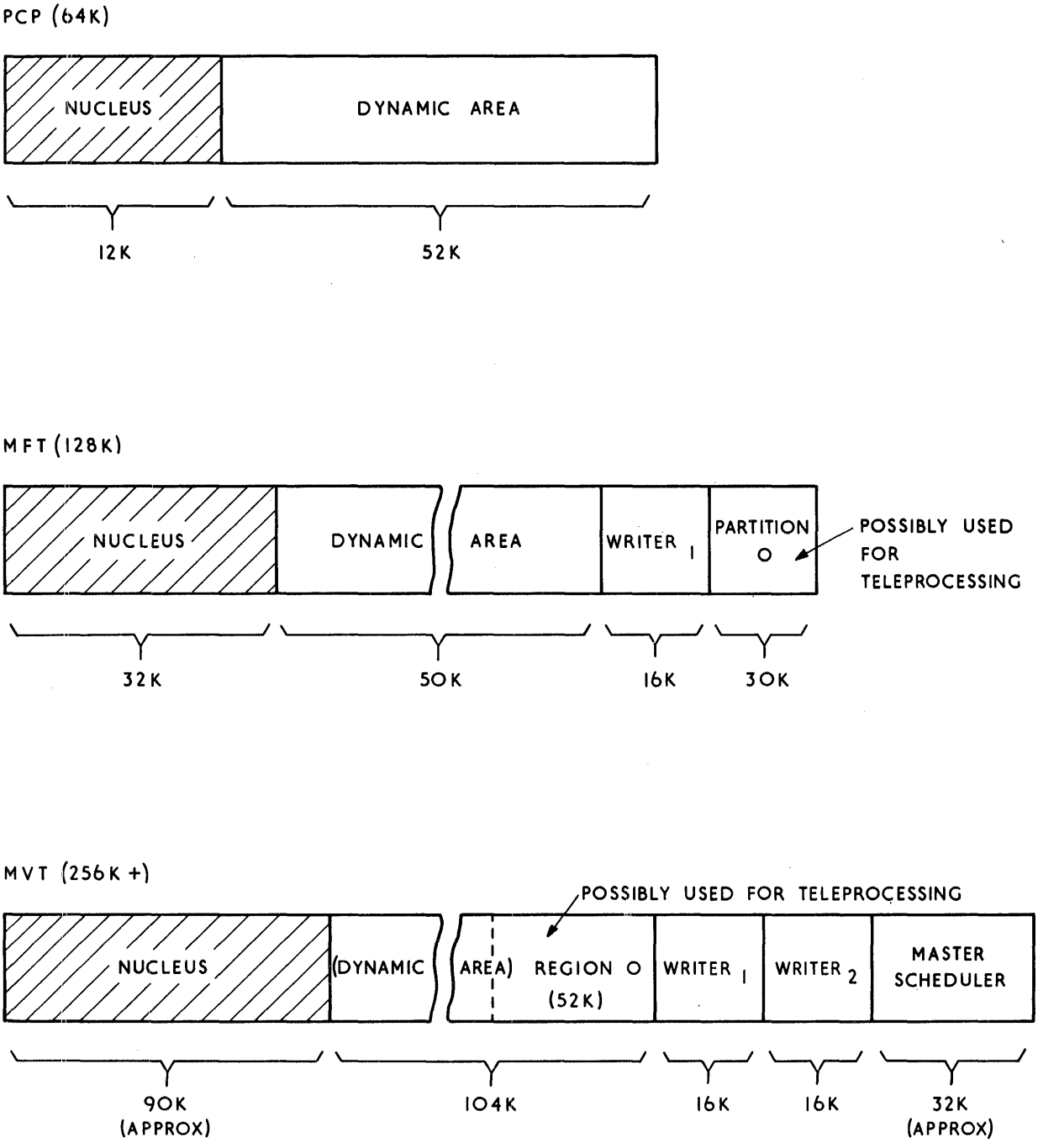
The use of certain operating system features requires additional main storage and resources. A table listing the minimum requirements of such features is given below (Figure D-2).

Facilities	Control program	Main storage required	System/360 model	Resources required
Batched processing	PCP	64K	30F	1. Central processing unit Operator's console Card reader punch Direct-access storage device Printer
	MFT	128K	40G	2. As for 1
	MVT	256K	40H	3. As for 1, with interval timer
Teleprocessing	PCP	Not available; QTAM requires MFT or MVT		
	MFT	128K (see Fig.D-3)	40G	4. As for 2, with Direct-access storage device for message queues Communication terminals with adapter or control units
	MVT	256K (see Fig.D-3)	40H 50I	5. As for 4, with interval timer
Batched processing with multiple console support (MCS)	PCP	Not available; MCS requires MFT or MVT		
	MFT	128K	40G	6. As for 2, with up to 31 extra operator's consoles
	MVT	256K	40H	7. As for 6
Multiprocessing	MVT	2x256K	2x40H	8. As for 3 except that two CPUs and two operator's consoles are required

Notes on resources:

1. Direct-access storage devices for PCP, MFT, and MVT.
2. Printer: 1403 and 1443, with chain (HN, PN, or QN) or bar (S2H or 63-character).
3. Character set: EBCDIC or USACII-8.
4. Operator's console: 1052 printer keyboard model 7 (with dual case printing element) with 2150 console or with 1052 adapter; composite consoles currently supported; 2250 console; model 85 integrated operator's console 220C.

Figure D-2. Minimum System Requirements



Note: the MFT and MVT diagram show the possible configurations with the teleprocessing facility. As teleprocessing requires a minimum of two partitions or regions, the space available in the dynamic area may not be sufficient with the minimum configuration.

Figure D-3. Possible Minimum Configurations of Main Storage

Appendix E: PL/I Library Subroutines

The following list comprises all the library modules provided for Version 5 of the PL/I (F) compiler. It gives the length in bytes of each module; and a brief description of its function. All the modules are resident in system libraries in auxiliary storage, either in the PL/I subroutine library (SYS1.PL1LIB), or in the link library (SYS1.LINKLIB). Those modules marked * reside in the link library.

Module	Length	Description
IHEABN	12	ABEND option
IHEABU	184	ABS (complex fixed-point binary)
IHEABV	544	ABS (complex fixed-point decimal)
IHEABW	128	ABS (complex short floating-point)
IHEABZ	128	ABS (complex long floating-point)
IHEADD	216	ADD (real fixed-point decimal)
IHEADV	96	ADD (complex fixed-point decimal)
IHEAPD	360	Conversion (real fixed-point decimal)
IHEATL	536	ATAN and ATAND (real long floating-point)
IHEATS	408	ATAN and ATAND (real short floating-point)
IHEATW	304	ATAN and ATANH (complex short floating-point)
IHEATZ	296	ATAN and ATANH (complex long floating-point)
IHEBEG	136	Messages to operator
IHEBSA	296	AND (byte-aligned bit strings)
IHEBSC	272	Comparison (byte-aligned bit strings)
IHEBSD	192	Comparison (bit strings, any alignment)
IHEBSF	480	BOOL
IHEBSI	296	INDEX (bit strings)
IHEBSK	472	Bit-string (assignment, concatenation, REPEAT)
IHEBSM	384	Bit-string (assignment, padding)
IHEBSN	192	NOT (byte-aligned bit string)
IHEBSO	312	OR (byte-aligned bit strings)
IHEBSS	240	SUBSTR (bit string)
IHEBST	512	TRANSLATE (bit string)
IHEBSV	408	VERIFY (bit string)
IHECFA	160	ONLOC

	Module	Length	Description
	IHECFB	576	ONCODE
	IHECFC	88	ONCOUNT
	IHECKP	184	Checkpoint (CALL IHECKPT)
*	IHECLS	922	Closing files (Versions 1, 2, and 3 only)
*	IHECLT	1298	Closing files
	IHECNT	72	COUNT/LINENO
	IHECSC	200	Comparison (two character strings)
	IHECSI	168	INDEX (character string)
	IHECSK	320	Character string (concatenation, REPEAT)
	IHECSM	280	Character string (assignment, HIGH, LOW)
	IHECSS	224	SUBSTR (character string)
	IHECST	304	TRANSLATE (character string)
	IHECSV	198	VERIFY (character string)
*	IHECTT	1718	Closing files
	IHEDBN	344	Conversion (bit string to arithmetic)
	IHEDCN	495	Conversion (character string to arithmetic)
	IHEDDI	1248	Stream-oriented transmission
	IHEDDJ	448	Array handling
	IHEDDO	648	Data-directed output
	IHEDDP	640	Array handling
	IHEDDT	760	Data-directed output
	IHEDIA	584	Edit-directed input
	IHEDIB	280	Edit-directed input
	IHEDID	448	Edit-directed input
	IHEDIE	456	Edit-directed input
	IHEDIL	48	Edit-directed input
	IHEDIM	528	Edit-directed input
	IHEDMA	248	Conversion (arithmetic data)
	IHEDNB	248	Conversion (arithmetic to bit string)
	IHEDNC	632	Conversion (arithmetic to character string)
	IHEDOA	224	Edit-directed output
	IHEDOB	328	Edit-directed output
	IHEDOD	296	Edit-directed output

	Module	Length	Description
	IHEDOE	224	Edit-directed output
	IHEDOM	584	Edit-directed output
	IHEDSP	612	Messages to operator
	IHEDUM	280	Main storage dump (CALL IHEDUMC, IHEDUMJ, IHEDUMP, IHEDUMT)
	IHEDVU	408	DIVIDE (complex fixed-point binary)
	IHEDVV	576	DIVIDE (compled fixed-point decimal)
	IHEDZW	104	Division (complex short floating-point)
	IHEDZZ	104	Division (complex long floating-point)
	IHEEFL	736	ERF, ERFC (real long floating-point)
	IHEEFS	384	ERF, ERFC (real short floating-point)
*	IHEERD	720	Execution-time error messages
*	IHEERE	1704	Execution-time error messages
*	IHEERI	896	Execution-time error messages
*	IHEERN	4096	Execution-time error messages (Versions 1 and 2 only)
*	IHEERO	856	Execution-time error messages
*	IHEERP	1208	Execution-time error messages
	IHEERR	1816	Error and interrupt handling
*	IHEERS	848	Execution-time messages (SNAP)
*	IHEERT	712	Execution-time error messages
*	IHEESM	1768	Execution-time messages (SNAP, CHECK)
*	IHEESS	1960	Execution-time messages (SNAP, CHECK: Version 2 only)
	IHEEXL	456	EXP (real long floating-point)
	IHEEXS	256	EXP (real short floating-point)
	IHEEXW	136	EXP (complex short floating-point)
	IHEEXZ	136	EXP (complex long floating-point)
	IHEHTL	272	ATANH (real long floating-point)
	IHEHTS	192	ATANH (real short floating-point)
	IHEIBT	576	Stream-oriented output (PUT)
	IHEIGT	1340	Record-oriented transmission (multitasking)
	IHEINT	436	Record-oriented transmission (multitasking)
	IHEIOA	360	Stream-oriented input (GET)
	IHEIOB	480	Stream-oriented output (PUT)
	IHEIOC	288	Stream-oriented transmission (STRING)

	Module	Length	Description
	IHEIOD	640	Edit-directed transmission
	IHEIOE	176	Implicit OPEN (Versions 1,2,3 only)
	IHEIOF	736	Stream-oriented input
	IHEIOG	1104	Record-oriented transmission
	IHEIOH	200	BDAM/BISAM housekeeping (Version 2 only)
*	IHEIOJ	1992	BSAM housekeeping (Versions 2,3 only)
	IHEION	248	Record I/O transmitter
	IHEIOP	488	PAGE option/format
	IHEIOX	328	SKIP option
*	IHEITB	3772	BSAM/CONSECUTIVE or REGIONAL UNBUFFERED interface
*	IHEITC	2604	BSAM/REGIONAL SEQUENTIAL (output) interface
*	IHEITD	2270	QISAM/INDEXED SEQUENTIAL interface
*	IHEITE	1760	BISAM/INDEXED DIRECT interface
*	IHEITF	1845	BDAM/REGIONAL DIRECT interface
*	IHEITG	1122	QSAM/CONSECUTIVE RECORD BUFFERED interface
*	IHEITH	2610	BISAM/INDEXED DIRECT (multitasking) interface
*	IHEITJ	2650	BDAM/REGIONAL DIRECT (multitasking) interface
*	IHEITK	622	QSAM/READ spanned records
*	IHEITL	492	QSAM/WRITE spanned records
*	IHEITP	874	QTAM/teleprocessing interface
	IHEJXI	320	Initialize and address arrays
	IHEJXS	104	Finds first and last elements of array
	IHEKCA	1560	Checks decimal picture specifications
	IHEKCB	1464	Checks sterling picture specifications
	IHEKCD	256	Checks character picture specifications
	IHELDI	2072	List-directed input director
	IHELDO	1048	List-directed output director
	IHELNL	360	LOG(x) (real long floating-point)
	IHELNS	256	LOG(x) (real short floating-point)
	IHELNW	268	LOG(z) (complex short floating-point)
	IHELNZ	288	LOG(z) (complex long floating-point)
	IHELSP	1064	List processing storage management
	IHEM91	344	Model 91 execution-time error routines

	Module	Length	Description
	IHEMAI	8	Addresses IHEBEG if no main procedure
	IHEMPU	240	MULTIPLY (complex fixed binary)
	IHEMPV	288	MULTIPLY (complex fixed decimal)
	IHEMSI	32	'STIMER unavailable' message
	IHEMST	32	'TIMER unavailable' message
	IHEMSW	136	I/O event counter (multitasking)
	IHEMXB	96	MAX (real fixed-point binary)
	IHEMXD	120	MAX (real fixed-point decimal)
	IHEMXL	96	MAX (real long floating-point)
	IHEMXS	96	MAX (real short floating-point)
	IHEMZU	240	z_1*z_2 (complex fixed-point binary
	IHEMZV	672	z_1*z_2 (complex fixed-point decimal)
	IHEMZW	64	z_1*z_2 (complex short floating-point)
	IHEMZZ	64	z_1*z_2 (complex long floating-point)
	IHENL1	280	ALL or ANY for simple array
	IHENL2	192	ALL or ANY for interleaved array
	IHEOCL	1338	Explicit/implicit open/close (non-multitasking)
	IHEOCT	2190	Explicit/implicit open/close (multitasking)
*	IHEOPN	920	Implicit/explicit open routine
*	IHEOPO	1828	Implicit/explicit open routine
*	IHEOPP	1874	Implicit/explicit open routine
*	IHEOPQ	1296	Implicit/explicit open routine
*	IHEOPZ	992	Format director: REGIONAL DIRECT output
	IHEOSD	216	Obtain current date
	IHEOSE	80	Terminate current task abnormally
	IHEOSI	72	Invoke STIMER (with WAIT if required)
	IHEOSS	56	Terminate job step abnormally
	IHEOST	88	Obtain time of day
	IHEOSW	1060	EVENT counter (non-multitasking)
	IHEPDF	144	PROD (real fixed-point binary/decimal) interleaved array
	IHEPDL	88	PROD (real long floating-point) interleaved array
	IHEPDS	88	PROD (real short floating-point) interleaved array
	IHEPDW	120	PROD (complex short floating-point) interleaved array

Module	Length	Description
IHEPDX	272	PROD (complex binary or decimal) interleaved array
IHEPDZ	120	PROD (complex long floating-point) interleaved array
IHEPRT	656	COPY/error message via SYSPRINT
IHEPSF	160	PROD (real fixed-point binary/decimal) simple array
IHEPSL	72	PROD (real long floating-point) simple array
IHEPSS	72	PROD (real short floating-point) simple array
IHEPSW	96	PROD (complex short floating-point) simple array
IHEPSX	256	PROD (complex binary or decimal) simple array
IHEPSZ	96	PROD (complex long floating-point) simple array
IHEPTT	768	COPY/error message via SYSPRINT (multitasking)
IHERES	100	Checkpoint/restart interface
IHESAP	2488	Storage management (non-multitasking)
IHESHL	248	SINH (real long floating-point)
IHESHS	192	SINH (real short floating-point)
IHESIZ	16	Length of PRV in Register 1
IHESMF	136	SUM (real binary or decimal) interleaved array
IHESMG	128	SUM (real short floating-point) interleaved array
IHESMH	128	SUM (real long floating-point) interleaved array
IHESMX	224	SUM (complex binary or decimal) interleaved array
IHESNL	416	SIN (real long floating-point)
IHESNS	320	SIN (real short floating-point)
IHESNW	320	SIN (complex short floating-point)
IHESNZ	368	SIN (complex long floating-point)
IHESQL	160	SQRT (real long floating-point)
IHESQS	168	SQRT (real short floating-point)
IHESQW	152	SQRT (complex short floating-point)
IHESQZ	144	SQRT (complex long floating-point)
IHESRC	344	Passes data to condition functions
IHESRD	128	Passes data to ONKEY function
IHESRT	1348	Sort/merge interface
IHESSF	168	SUM (real fixed-point binary/decimal) simple array
IHESSG	104	SUM (real short floating-point) simple array
IHESSH	104	SUM (real long floating-point) simple array

	Module	Length	Description
	IHESSX	216	SUM (complex fixed-point binary/decimal) simple array
	IHESTA	1124	Abnormal termination error message
	IHESTG	1108	Length of structure string
	IHESTP	1432	Assigns bit/character string to string pseudo-variable
	IHESTR	1592	Structure address and length
*	IHESUB	16	Subtask initialization interface
	IHETAB	16	Default table for tabulation
	IHETCV	208	Controlled variable storage (multitasking)
	IHETEA	248	Event variable assignment
	IHETER	272	ON-field search (multitasking)
	IHETEV	240	COMPLETION pseudo-variable
*	IHETEX	1464	Active task terminated message
	IHETHL	280	TANH (real long floating-point)
	IHETHS	200	TANH (real short floating-point)
	IHETNL	344	TAN (real long floating-point)
	IHETNS	272	TAN (real short floating-point)
	IHETNW	184	TAN (complex short floating-point)
	IHETNZ	184	TAN (complex long floating-point)
*	IHETOM	493	Operator messages
	IHETPB	56	PRIORITY built-in function
	IHETPR	268	PRIORITY pseudo-variable
	IHETSA	5720	Object program management (multitasking)
	IHETSE	88	Abnormally terminates current task
	IHETSS	72	Terminates PL/I program abnormally (multitasking)
	IHETSW	1520	EVENT counter (multitasking)
	IHEUPA	192	Complex data item
	IHEUPB	232	Complex numeric field
	IHEVCA	272	Defines attributes of arithmetic data in character form
	IHEVCS	480	Conversion director (complex data-string)
	IHEVFA	232	Radix conversion: binary to decimal
	IHEVFB	224	Conversion (long-float to fixed-point binary)
	IHEVFC	40	Conversion (long-float to floating-point variable)
	IHEVFD	88	Conversion (fixed-point binary to long floating-point)

Module	Length	Description
IHEVFE	32	Conversion (float-point to long floating-point)
IHEVKB	736	Conversion (float-point decimal to packed decimal)
IHEVKC	720	Conversion (sterling to packed decimal)
IHEVKF	1504	Conversion (packed decimal to fixed/floating-point)
IHEVKG	1248	Conversion (packed decimal to sterling)
IHEVPA	352	Conversion (packed decimal to long floating-point)
IHEVPB	408	Conversion (packed decimal to F-format)
IHEVPC	492	Conversion (packed decimal to E-format)
IHEVPD	264	Conversion (packed decimal to decimal)
IHEVPE	616	Conversion (F-, E-format to packed decimal)
IHEVPF	72	Conversion (decimal to packed decimal)
IHEVPG	560	Conversion (binary fixed/floating-point to long floating-point)
IHEVPH	184	Conversion (bit string to long floating-point)
IHEVQA	208	Conversion (floating-point to binary)
IHEVQB	1004	Conversion (decimal constant to coded arithmetic)
IHEVQC	600	Conversion (coded arithmetic to F-, E-format/character string)
IHEVSA	320	Assignment (bit string to bit string)
IHEVSB	208	Conversion (bit string to character string)
IHEVSC	176	Assignment (character string to character string)
IHEVSD	416	Conversion (character string to bit string)
IHEVSE	352	Assignment (character string to picture string)
IHEVSF	240	Conversion (bit string to picture string)
IHEVTB	136	Float-point table for conversions
IHEXIB	88	X**n (binary to integer)
IHEXID	136	X**n (decimal to integer)
IHEXIL	152	X**n (long floating-point to integer)
IHEXIS	152	X**n (short floating-point to integer)
IHEXIU	120	Z**n (complex binary to integer)
IHEXIV	192	Z**n (complex decimal to integer)
IHEXIW	256	Z**n (complex short floating-point to integer)
IHEXIZ	256	Z**n (complex long floating-point to integer)
IHEXXL	152	X**Y (long floating-point)
IHEXXS	144	X**Y (short floating-point)

	Module	Length	Description
	IHEXXW	280	Z_1**Z_2 (complex short floating-point)
	IHEXXZ	280	Z_1**Z_2 (complex long floating-point)
	IHEYGF	432	POLY (binary or decimal)
	IHEYGL	240	POLY (long floating-point)
	IHEYGS	240	POLY (short floating-point)
	IHEYGW	280	POLY (complex short floating-point)
	IHEYGX	688	POLY (complex binary or decimal)
	IHEYGZ	280	POLY (complex long floating-point)
*	IHEZZA	1296	SNAP dump index (I) (Version 3 only)
*	IHEZZB	1704	SNAP dump index (II) (Version 3 only)
*	IHEZZC	2960	SNAP dump control routine
*	IHEZZF	1596	Save-area trace for SNAP

Appendix F: Shared Library

Introduction

The shared library feature enables certain preselected PL/I subroutine library modules to be shared between two or more PL/I programs being executed under the control of an MFT or MVT control program of the operating system.

The preselected group of modules is made resident in main storage (in the resident access method area for MFT, and in the link-pack area for MVT).

To create a shared library you must specify the modules that you require when you generate your operating system. You do this by means of a system generation macro instruction. The system generation process will link-edit these modules together with a transfer vector module, into a load module which is given the entry-point name IHELTVA, and stored in auxiliary storage in the link library, SYS1.LINKLIB.

The load module can be subsequently loaded permanently into main storage during initial program load. This load module is a reentrant load module and your installation must be able to make this type of module resident during initial program load[1].

To execute a PL/I program using the shared library, you must ensure that a dummy transfer vector module is link-edited to your program in order to resolve all references to the shared library modules. You do this by specifying the entry-point name IHELTTA for your PL/I program. This dummy transfer vector module is also created during system generation and is also stored in auxiliary storage in the PL/I subroutine library, SYS1.PL1LIB.

You can use the shared library by either of two methods:

1. By using standard IBM-supplied cataloged procedures and overriding the link-edit and loader job steps.

2. By providing your own cataloged procedures.

These methods and their required coding are described below.

---

[1]For more information on making reentrant load modules resident during initial program load refer to IBM System/360 Operating System: System Generation, and IBM System/360 Operating System: System Programmer's Guide.

How to Create a Shared Library

The PL1LIB System Generation Macro Instruction

To create a shared library, you must specify the modules that you require when you generate your operating system. You do this by means of a system generation macro instruction, PL1LIB[2].

To facilitate selection of modules required for your shared library, the PL/I subroutine library modules have been divided into 36 groups. Each group comprises modules that have associated functions, as shown in Figure F-1.

You cannot specify modules in Group 1 because these are housekeeping, string function, or stream input/output modules that cannot be used in a shared library; they are termed 'non-shareable' modules.

Group 2 modules will be included automatically during system generation if you have specified a multitasking shared library, or group 3 modules if you have specified a non-multitasking shared library.

You can specify any of the remaining groups (Groups 4 through 36) in any combination to form a shared library to fit your needs. You should specify only frequently used groups of modules to make best use of the shared library feature.

---

[2] For more information on the PL1LIB macro instruction refer to IBM System/360 Operating System: System Generation

GROUP	MAIN FUNCTIONS OF GROUP
1	Non-shareable modules
2	Multitasking storage management
3	Non-multitasking storage mangement
4	Error handling (ON-units)
5	List processing and structure mapping
6	Basic conversion package
7	Edit conversions
8	Complex conversions
9	Bit-string conversions
10	Character-string conversions
11	Picture conversions
12	Sterling conversions
13	Optimization=1 special conversions
14	Bit-string built-in functions
15	Character-string built-in functions
16	STRING built-in function and pseudo-variable
17	Real non-interleaved arrays
18	Real interleaved arrays
19	Complex non-interleaved arrays
20	Complex interleaved arrays
21	Real arithmetic operators
22	Complex arithmetic operators
23	Real short arithmetic built-in functions
24	Real long arithmetic built-in functions
25	Complex short artihmetic built-in functions
26	Complex long arithmetic built-in functions
27	Non-multitasking data-directed I/O
28	Non-multitasking list-directed I/O
29	Non-multitasking edit-directed I/O
30	Multitasking data-directed I/O
31	Multitasking list-directed I/O
32	Multitasking edit-directed I/O
33	Non-multitasking record I/O
34	Multitasking record I/O
35	Non-multitasking record I/O wait
36	Multitasking record I/O wait

Figure F-1. Shared-Library Module Groups

Initial Program Load (IPL)

Once created, your shared library load module is given the entry-point name IHELTVA and is stored in auxiliary storage in the link library, SYS1.LINKLIB. To store this load module permanently in main storage you must first build a new reentrant load module user list for it in the parameter library, SYS1.PARMLIB; and second, you must ensure that the use of this list by the system is specified during initial program load. You can give this user list a name of the form IEAIGGxx, where xx are any two digits other than 00. The following example of coding suggests how you create such a list by means of the IEBUPDTE[1] utility program; the method of specifying its use by the system is given in the following paragraph. For illustration purposes, the example has been given the user list name IEAIGG01.

```
//BLDLIST  EXEC   PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD     SYSOUT=A
//SYSUT2   DD     DSN=SYS1.PARMLIB,
//                UNIT=2314,
//                VOL=(,RETAIN,SER=MVT111),
//                DISP=OLD
//SYSIN    DD     *
./         ADD    NAME=IEAIGG01,LIST=ALL
./         NUMBER NEW=01,INCR=02
     SYS1.LINKLIB  IHELTVA
./    ENDUP
/*
```

---

[1]For more information about the IBM utility program IEBUPDTE refer to IBM System/360 Operating System: Utilities

The ADD statement identifies your shared library load module, IHELTVA, in the new reentrant load module list (IEAIGG01).[1] When the nucleus initialization program (NIP) requests the system parameters, the operator reponds by specifying 'RAM=01' (the digits 01 correspond to the last two digits of the user list name IEAIGG01). This response causes your load module to be loaded into the relevant area of main storage.

If you require both the standard and shared library load module lists, you must instruct the operator to respond to the NIP request for system parameters by specifying 'RAM=00,01'. You should note the following points:

- If your installation standard reentrant load module list (IEAIGG00) refers to SYS1.LINKLIB and not SYS1.SVCLIB, you can add your module entry-point name (IHELTVA) to this list. In this case you do not need to create a new procedure library list; your shared-library load module will be automatically stored in main storage at initial program load.

- If your installation standard reentrant load module list (IEAIGG00) does not refer to SYS1.LINKLIB, your system must have been generated with the ability to communicate with the operator during initial program load. This is because the operator must specify the use of the alternative list as described above.

How to Use a Shared Library

When executing programs using the shared library you must ensure that the first module to be processed by the linkage editor or loader is the dummy transfer vector module IHELTTA. Also ensure that the entry-point name of the resulting load module is specified as IHELTTA. You should note the following points:

- You can have a shared library for either non-multitasking or multitasking programs. You cannot, however, use a multitasking program with a non-multitasking shared library and conversely, you cannot use a non-multitasking program with a multitasking shared library. You should find out whether your installation has a multitasking or non-multitasking shared library.

- A load module created for use with one shared library will not execute with a different shared library. You will have to link-edit the object module again, including the dummy transfer vector module IHELTTA for the different shared library.

- You must remember that the linkage editor or loader require a large amount of main storage for external symbol dictionary tables while processing the dummy transfer vector module IHELTTA. If you specify SIZE=200K in the PARM field of your EXEC statement for the linkage editor or loader (and use a region or partition of equivalent size), you will get sufficient main storage for processing with the largest possible shared library.

- Your PL/I program may take slightly longer to execute when using a shared library, because all library calls have to pass through the transfer vectors. However, your main storage requirements for a region will be greatly reduced if you have carefully selected your shared library modules to suit the operating environment.

- The address of your shared library module in the link-pack area of main storage is found by IHELTTA issuing a LOAD macro instruction. If the shared-library load module IHELTVA was not made resident in main storage at initial program load, and you try to run your program with shared-library job control statements, the shared-library module will be loaded into the region or partition. Your job will still execute, but you will need to specify a larger region size(MVT), or use a larger partition (MFT).

---

[1] For more information about reentrant load module lists refer to IBM/360 Operating System: System Programmer's Guide.

USING STANDARD IBM CATALOGED PROCEDURES

In this method you make use of standard IBM-supplied cataloged procedures that contain a link-edit job step or that use the linkage loader. If you use this method you must override certain statements in the link-edit or loader job step of the cataloged procedure to ensure that the shared-library transfer module (IHELTTA) is the first module to be included by the linkage editor or loader, and that its entry point in the resulting load module has the name IHELTTA.

The methods for doing this for each individual cataloged procedure differ slightly and are described below.

Cataloged Procedures using the Linkage Editor

Compile and Link-Edit (PL1LFCL)

You may use this cataloged procedure by reversing the order of concatenation of the data sets defined in the DD statement with the name SYSLIN. This ensures that the first module specified is IHELTTA. The suggested method of doing this is as follows:

```
//S           EXEC    PL1LFCL
//LKED.SYSLIN DD      DDNAME=LIN
//LKED.OBJ    DD      DSN=&&LOADSET,DISP=(OLD,DELETE)
//LKED.LIN    DD      *
  INCLUDE     SYSLIB(IHELTTA)
  INCLUDE     OBJ
  ENTRY       IHELTTA

              (add further input here)

/*
```

You can add other linkage-editor control statements by placing them as indicated. If, for example, you wish to give the resulting load module the name MINE, you should add the statement:

```
 NAME     MINE(R)
```

between the ENTRY and /* statements.

Note: You must not use DDNAME=SYSIN in the LKED.SYSLIN DD statement as this would be concatenated with an identical DD statement provided by the cataloged procedure and cause the input stream to be processed twice.

Compile, Link-Edit and Execute (PL1LFCLG)

This cataloged procedure can be modified for use with a shared library in exactly the same way as described for PL1LFCL above.

Link-Edit and Execute (PL1LFLG)

You may use this cataloged procedure by specifying the compiled object module as the member DECK of the partitioned data set PL1.OBJECT, as follows:

```
//S           EXEC    PL1LFLG
//LKED.OBJ    DD    DSN=PL1.OBJECT(DECK),DISP=SHR
//LKED.SYSIN  DD    *
  INCLUDE     SYSLIB(IHELTTA)
  INCLUDE     OBJ
  ENTRY       IHELTTA


              (add further input here)


/*
```

Note: Since PL1LFLG does not specify a concatenated data set for the SYSLIN DD statement of the LKED step, but only specifies DDNAME=SYSIN, you can specify all the link-edit control statements by the LKED.SYSIN DD * statement.

Cataloged Procedures using the Linkage Loader

Compile, Load-and-Execute (PL1LFCG)

You may use this cataloged procedure by overriding the PARM parameter of the GO step to add the specified entry-point IHELTTA. You must also override the SYSLIN DD statement to permit the transfer vector module IHELTTA to be processed by the compiler object module. The suggested method for doing this is as follows:

```
//S           EXEC  PL1LFCG,PARM.GO='MAP,PRINT,EP=IHELTTA'
//GO.SYSLIN   DD    DSN=SYS1.PL1LIB(IHELTTA),DISP=SHR
//             DD    DSN=&&LOADSET,DISP=(OLD,DELETE)
```

Load-and-Execute (PL1LFG)

You may use this cataloged procedure by specifying the compiled object module as the member DECK of the partitioned data set PL1.OBJECT, as follows:

```
//S           EXEC  PL1LFG,PARM.GO='MAP,PRINT,EP=IHELTTA'
//GO.SYSLIN   DD    DSN=SYS1.PL1LIB(IHELTTA),DISP=SHR
//             DD    DSN=PL1.OBJECT(DECK),DISP=SHR
```

PROVIDING YOUR OWN CATALOGED PROCEDURES

You can provide your own cataloged procedures for use with a shared library. The following examples suggest how you might write these, basing them on the standard IBM-supplied cataloged procedures.

For illustration purposes, the names PL1LSCL, PL1LSCLG, etc., have been chosen, but you could use any names you choose. Once you have written your cataloged procedures you add them to the procedure library (SYS1.PROCLIB) at your installation by using the IBM utility program IEBUPDTE.

Cataloged Procedures using the Linkage Editor

Compile and Link-Edit (PL1LSCL)

When you write this cataloged procedure, you should remember that it needs a special member in the procedure library to hold the input control data to the linkage editor. In the example, this member is called PL1LSLD and includes the following linkage editor statements:

```
INCLUDE     SYSLIB(IHELTTA)
ENTRY       IHELTTA
```

Your cataloged procedure might look like this:

```
//PL1L       EXEC  PGM=IEMAA,PARM='LOAD,NODECK',REGION= 52K
//SYSPRINT   DD    SYSOUT=A
//SYSLIN     DD    DSN=&&LOADSET,DISP=,MOD,PASS.,UNIT=SYSDA,
//                 SPACE=(80,(250,100))
//SYSUT3     DD    DSN=&&SYSUT3,UNIT=SYSDA,SPACE=(80,(250,250)),
//                 DCB=BLKSIZE=80
//SYSUT1     DD    DSN=&&SYSUT1,SPACE=(1024,(60,60),,CONTIG),
//                 UNIT=SYSDA,SEP=(SYSUT3,SYSLIN),DCB=BLKSIZE=1024
//LKED       EXEC  PGM=IEWL,PARM='XREF,LIST',COND=(9,LT,PLIL),
//                 REGION=96K
//SYSLIB     DD    DSN=SYS1.PLILIB,DISP=SHR
//SYSLMOD    DD    DSN=&&GOSET(GO),UNIT=SYSDA,DISP=(MOD,PASS),
//                 SPACE=(1024,(50,20,1),RLSE)
//SYSUT1     DD    DSN=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(200,20)),
//                 SEP=(SYSLMOD,SYSLIB),DCB=BLKSIZE=1024
//SYSPRINT   DD    SYSOUT=A
//SYSLIN     DD    DSN=SYS1.PROCLIB(PL1LSLD),DISP=SHR
//           DD    DSN=&&LOADSET,DISP=(OLD,DELETE)
//           DD    DDNAME=SYSIN
```

Compile, Link-Edit and Execute (PL1LSCLG)

You can write this cataloged procedure in a similar way to that described above for PL1LSCL, remembering that again you require the special member PL1LSLD.

```
//PL1L       EXEC  PGM=IEMAA,PARM='LOAD,NODECK',REGION=52K
//SYSPRINT   DD    SYSOUT=A
//SYSLIN     DD    DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//                 SPACE=(80,(250,100))
//SYSUT3     DD    DSN=&&SYSUT3,UNIT=SYSDA,SPACE=(80,250,250)),
//                 DCB=BLKSIZE=80
//SYSUT1     DD    DSN=&&SYSUT1,SPACE=(1024,(60,60),,CONTIG),
//                 UNIT=SYSDA,SEP=(SYSUT3,SYSLIN),DCB=BLKSIZE=1024
//LKED       EXEC  PGM=IEWL,PARM='XREF,LIST',COND=(9,LT,PL1L),
//                 REGION=96K
//SYSLIB     DD    DSN=SYS1.PL1LIB,DISP=SHR
//SYSLMOD    DD    DSN=&&GOSET(GO),UNIT=SYSDA,DISP=(MOD,PASS),
//                 SPACE=(1024,(50,20,1),RLSE)
//SYSUT1     DD    DSN=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(200,20)),
//                 SEP=(SYSLMOD,SYSLIB),DCB=BLKSIZE=1024
//SYSPRINT   DD    SYSOUT=A
//SYSLIN     DD    DSN=SYS1.PROCLIB(PL1LSLD),DISP=SHR
//           DD    DSN=&&LOADSET,DISP=,OLD,DELETE.
//           DD    DDNAME=SYSIN
//GO         EXEC  PGM=*.LKED.SYSLMOD,COND=((9,LT,LKED),(9,LT,PL1L))
//SYSPRINT   DD    SYSOUT=A
```

Link-Edit and Execute (PL1LSLG)

Again you need the special member PL1LSLD in SYS1.PROCLIB.

```
//LKED      EXEC  PGM=IEWL,PARM='XREF,LIST',REGION=96K
//SYSLIB    DD    DSN=SYS1.PL1LIB,DISP=SHR
//SYSLMOD   DD    DSN=&&GOSET(GO),UNIT=SYSDA,DISP=(MOD,PASS),
//               SPACE=(1024,(50,20,1),RLSE)
//SYSUT1    DD    DSN=&&SYSUT1,SPACE=(1024,(200,20)),
//               UNIT=SYSDA,SEP=(SYSLMOD,SYSLIB),DCB=BLKSIZE=1024
//SYSPRINT  DD    SYSOUT=A
//SYSLIN    DD    DSN=SYS1.PROCLIB(PL1LSLD),DISP=SHR
//          DD    DDNAME=SYSIN
//GO        EXEC  PGM=*.LKED.SYSLMOD,=(9,LT,LKED)
//SYSPRINT  DD    SYSOUT=A
```

Cataloged Procedures using the Linkage Loader

In cataloged procedures that use the linkage loader, SYSLOUT is the ddname of the diagnostic message output data set. This name should have been specified at system generation; if it has not, you should specify PARM = 'NOPRINT' in your EXEC statement. If you do not specify this, and the loader was generated without the ddname SYSLOUT, the system gives the loader diagnostic message output data set the default name SYSPRINT, which is also the standard diagnostic message output data set name for PL/I programs. If both the loader and a PL/I program attempt to execute DD statements with this ddname under an MFT or MVT control program, you may lose your output or input/output errors may occur when an output writer attempts to print it.

You do not need the special member PL1LSLD in the procedure library because you must name the entry point of the resulting load module in the PARM parameter of your EXEC statement.

Compile, Load-and-Execute (PL1LSCG)

In this cataloged procedure, your compiled object module is stored in the temporary data set &&LOADSET which is concatenated with the transfer vector module IHELTTA.

```
//PL1L      EXEC  PGM=IEMAA,PARM='LOAD,NODECK',REGION=52K
//SYSPRINT  DD    SYSOUT=A
//SYSLIN    DD    DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//               SPACE=(80,(250,100))
//SYSUT3    DD    DSN=&&SYSUT3,UNIT=SYSDA,SPACE=(80,(250,250)),
//               DCB=BLKSIZE=80
//SYSUT1    DD    DSN=&&SYSUT1,SPACE=(1024,(60,60),,CONTIG),
//               UNIT=SYSDA,SEP=(SYSUT3,SYSLIN),DCB=BLKSIZE=1024
//GO        EXEC  PGM=LOADER,PARM='MAP,PRINT,EP=IHELTTA',
//               REGION=96K,COND=(9,LT,PL1L)
//SYSLIB    DD    DSN=SYS1.PL1LIB,DISP=SHR
//          DD    DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSLOUT   DD    SYSOUT=A
//SYSPRINT  DD    SYSOUT=A
```

Load-and-Execute (PL1LSG)

In this cataloged procedure, your compiled object module is the member DECK in the partitioned data set PL1.OBJECT, and you may define this in the GO step as follows:

```
//GO.LOADIN DD DSN=PL1.OBJECT(DECK),DISP=SHR
```

Note: You might use the ddname LOADIN instead of SYSIN to define input to the loader, because SYSIN may be required to define the PL/I (F) default input data set for the program being loaded.

Your cataloged procedure may look like this:

```
//GO          EXEC  PGM=LOADER,PARM='MAP,PRINT,EP=IHELTTA',
//                  REGION=96K
//SYSLIB      DD    DSN=SYS1.PL1LIB,DISP=SHR
//SYSLIN      DD    DSN=SYS1.PL1LIB(IHELTTA),DISP=SHR
//             DD    DDNAME=LOADIN
//SYSOUT      DD    SYSOUT=A
//SYSPRINT    DD    SYSOUT=A
```

Appendix G: IBM System/360 Model 91

Note: In the following discussion, the terms exception and interrupt are used. An exception is a hardware occurrence (such as an overflow error) which can cause a program interrupt. An interrupt is a suspension of normal program activities. There are many possible causes of interrupts, but the following discussion is concerned only with interrupts resulting from hardware exceptions.

IBM System/360 Models 91 and 195 are high-speed processing systems in which more than one instruction is executed concurrently. As a result, an exception may be detected and an interrupt occur when the address of the instruction which caused the exception is no longer held in the central processing unit. Consequently, the instruction causing the interrupt cannot be precisely identified. Interrupts of this type are termed imprecise. When an exception occurs, the machine stops decoding further instructions and ensures that all instructions which were decoded prior to the exception are executed before honoring the exception. Execution of the remaining decoded instructions may result in further exceptions occurring. An imprecise interrupt in which more than one exception has occurred is known as a multiple-exception imprecise interrupt.

The (F) compiler permits processing of imprecise interrupts only when the compiler option OBJIN is in effect. The option must always apply when the compiler is used to produce object programs for execution on a Model 91 or 195. The effect of the option is:

1. To cause the compiler to insert 'no-operation' instructions at certain points in the program to localize imprecise interrupts to a particular segment of the program, thus ensuring that interrupt processing results in the action specified in the source program. (A 'no-operation' instruction is an assembler language instruction of the form BCR x, 0 (where x is not equal to zero). This instruction is implemented in Models 91 and 195 in such a way that its execution is delayed until all previously decoded instructions have been executed.) 'No-operation' instructions are generated:

   a. Before an ON-statement.

   b. Before a REVERT statement.

   c. Before compiled code to set the SIZE condition.

   d. Before compiled code to change prefix options.

   e. For a null statement. (This feature provides the programmer with source language control over the timing of program interrupts.)

   f. Before every statement if the STMT compiler option applies. (This is an important debugging tool.)

Note: The average PL/I program can be executed on a Model 91 or 195 without significant degradation in execution time or storage requirements. However, the use of the STMT compiler option will cause both the execution time and the object-code storage requirements to increase in direct proportion to the number of statements in the program. The benefits obtained as a result of the debugging aid that this option gives should offset and justify these increases.

2. To create an external symbol dictionary (ESD) entry for the PL/I library module IHEM91 (required only when an object program is to be executed on a Model 91 or 195). The module is included when the object module is link-edited, and it is called when an imprecise interrupt is detected. Module IHEM91 provides the facilities for:

   a. Detecting multiple-exception imprecise interrupts.

   b. Setting the value that is returned by the ONCOUNT built-in function.

   c. Raising the appropriate PL/I conditions.

The order of processing the exceptions is as follows:

1. PL/I conditions in the order:

   UNDERFLOW

   FIXEDOVERFLOW or SIZE

   ERROR if system action is required for either FIXEDOVERFLOW or SIZE

OVERFLOW

ERROR if system action is required for OVERFLOW

ZERODIVIDE

ERROR if system action is required for ZERODIVIDE.

Note: The conditions FIXEDOVERFLOW and SIZE cannot occur together, since the same hardware condition raises both of them.

2. Hardware exceptions in the order:

   data

   specification

   addressing

   protection

Conditions and exceptions are raised in the above order until one of the following situations occurs:

1. A GO TO statement in an ON-unit is executed. All other exceptions will then be lost.
2. The ERROR condition is raised. If the program is terminated as a result of this action (i.e., system action causing the ERROR condition to be raised, followed by the FINISH condition), messages will be printed to indicate the nature of the unprocessed exceptions. The exceptions themselves will not be processed.

When an interrupt results from multiple exceptions, only one of the PL/I conditions is raised for each type of exception that has occurred.

When a multiple-exception imprecise interrupt occurs, the ONCOUNT built-in function provides a binary integer count of the number of exception types, including the current one, remaining to be processed. (The count does not include PL/I ON-conditions.) If the ONCOUNT built-in function is used when only a single exception has occurred, or if it is used outside an ON-unit, a count value of zero is indicated.

Programs compiled with the OBJIN compiler option can be executed on other IBM System/360 models supported by the IBM System/360 Operating System.

Appendix H: Compiler Data Sets

DATA SET	AUXILIARY STORAGE REQUIRED(Approximate in that considerable variation may result from the nature of individual source programs)					
SYSPUNCH	80 * S bytes					
SYSLIN	80 * S bytes					
SYSUT1	S	PL/I (F) Operating in	NUMBER OF TRACKS REQUIRED			
			2301 Drum Storage	2311 Disk Storage	2314 Disk Drive	2321 Data Cell
	150	44K	4	8	4	22 (24)
		100K	0	0	0	0
		200K	0	0	0	0
	500	44K	14	31	16	92 (99)
		100K	9 (11)	18 (22)	9	27 (33)
		200K	0	0	0	0
	1000	44K	28	64	32	192 (202)
		100K	34 (37)	68 (74)	34	102 (111)
		200K	9 (12)	15 (20)	6	27 (28)
	Numbers in parentheses are for the EXTDIC option. Since it is not advisable to use this option if SIZE=44K, the figures for this value must be taken as applying to a 48K main storage size. Source programs which are large enough to cause the SYSUT1 data set to be used will compile more efficiently if this data set is on a drum or in a contiguous area on a disk.					
SYSUT3	2 * R * P bytes — 48-character processing T bytes — Compile-time processing 2 * T bytes — Compile-time and 48-character processing					
SYSPRINT	120 * (Q + R + 10*S + U + U + 30) bytes └──if EXTREF option specified └──if XREF option specified └──if ATR option specified └──if LIST option specified └──if SOURCE option specified └──if SOURCE2 option specified					

Key:

P = Average length of record (in bytes).

Q = Number of input records (records containing source text + records included through the compile-time INCLUDE statement).

R = Number of records containing source statements after compile-time processing completed.

S = Number of source statements after compile-time processing completed.

T = Size (in bytes) of program after compile-time processing.

U = Number of variables in the program.

Appendix I: ON, Return, and User Completion Codes

ON-Codes

The ONCODE built-in function may be used by the programmer in any ON-unit to determine the nature of the error or condition which caused entry into that ON-unit.

An ON-unit, which has been established by the execution of an ON-statement, is entered when the associated ON-condition is raised during execution of PL/I compiled code or of a PL/I library module. Thus, for example, a FIXEDOVERFLOW ON-unit would be entered whenever any of the conditions occur for which the language demands the raising of the FIXEDOVERFLOW condition. Two ON-conditions, ERROR and FINISH, require special explanation.

The ERROR condition is raised:

1. Upon execution of a SIGNAL ERROR statement.
2. As a result of system action for those ON-conditions for which the language specifies system action to be 'comment and raise the ERROR condition.'
3. As a result of an error (for which there is no ON-condition) occurring during program execution.

The FINISH condition is raised:

1. Upon execution of a SIGNAL FINISH,STOP, or EXIT statement.
2. Upon normal completion of the MAIN procedure of a PL/I program.
3. Upon completion of the action associated with the raising of the ERROR condition, except when a GO TO statement in the ON ERROR unit has resulted in transfer of control out of that unit.

As a general rule, the value of the code returned by the ONCODE built-in function is that of the specific condition which caused entry into the ON-unit. Thus, in an ON CONVERSION unit, the programmer can expect an ON-code corresponding to one of the conversion conditions which cause the CONVERSION condition to be raised in PL/I. However, this is not necessarily true when executing an ON-error or an ON FINISH unit; the values are as follows:

1. When entered as a result of a SIGNAL ERROR or a SIGNAL FINISH, STOP, or EXIT statement, or as a result of normal termination, the ON-codes will be those of ERROR or FINISH respectively.
2. When entered for any other reason, the ON-code will be that associated with the error or condition which originally caused the ERROR condition to be raised.

Several separate but related occurrences may cause a particular PL/I ON-condition to be raised. For example, the TRANSMIT condition may be raised:

1. By execution of a SIGNAL TRANSMIT statement.
2. By occurrence of an input TRANSMIT error.
3. By occurrence of an output TRANSMIT error

Although it is often useful to know precisely what caused an ON-condition to be raised, at times it will be sufficient simply to know which ON-condition was raised. This will apply particularly if the ONCODE built-in function is used in an ERROR ON-unit after system action has occurred for an ON-condition. The ON-codes have therefore been grouped, each group containing the codes associated with a particular ON-condition.

From time to time it may become necessary or desirable to add new ON-codes into a group. Perhaps a group containing one ON-code only may be expanded. This fact must be remembered when the ONCODE built-in function is used to determine if a particular PL/I ON-condition has been raised. It is important to test to see whether the ON-code is within the range specified, even if there is only one ON-code in the range; otherwise, when a new set of library modules is used, it may become necessary to recompile the program.

The ON-codes and their associated conditions and errors are shown below. The groups and their ON-code ranges are shown in Figures I-1 and I-2. (Language ON-conditions are shown in capitals, others in lower case letters.)

Range	Group
3-5	As for 1000-9999
10-199	I/O ON-conditions
300-399	Computational ON-conditions
500-549	program check-out conditions
600-899	Conversion conditions
1000-9999	Error conditions (also 3-5)

Figure I-1. Main ON-Code Groupings

Range	Group
0	ONCODE
3	Source program error
4	FINISH
9	ERROR
10-19	NAME
20-39	RECORD
40-49	TRANSMIT
50-69	KEY
70-79	ENDFILE
80-89	UNDEFINEDFILE
90-99	ENDPAGE
100-299	(Unallocated)
300-309	OVERFLOW
310-319	FIXEDOVERFLOW
320-329	ZERODIVIDE
330-339	UNDERFLOW
340-349	SIZE
350-359	STRINGRANGE
360-369	AREA
370-499	(Unallocated)
500-509	CONDITION
510-519	CHECK
520-529	SUBSCRIPTRANGE
530-599	(Unallocated)
600-899	CONVERSION
900-999	(Unallocated)
1000-1199	I/O errors
1200-1499	(Unallocated)
1500-1699	Data processing errors
1700-1999	(Unallocated)
2000-2099	Unacceptable statement errors
2100-2999	(Unallocated)
3000-3499	Conversion errors
3500-3799	(Unallocated)
3800-3899	Structure and array errors
3900-3999	Tasking errors
4000-8090	(Unallocated)
8091-8199	Program interrupt errors
8200-8999	(Unallocated)
9000-999	System errors

Figure I-2. Detailed ON-Code Groupings

Code	Condition/Error
0	ONCODE function used out of context
3	Source program error
4	FINISH
9	ERROR
10	NAME
20	RECORD (signaled)
21	RECORD (record variable smaller than record size)
22	RECORD (record variable larger than record size)
23	RECORD (attempt to write zero length record)
24	RECORD (zero length record read)
40	TRANSMIT (signaled)
41	TRANSMIT (output)
42	TRANSMIT (input)
50	KEY (signaled)
51	KEY (keyed record not found)
52	KEY (attempt to add duplicate key)
53	KEY (key sequence error)
54	KEY (key conversion error)
55	KEY (key specification error)
56	KEY (keyed relative record/track outside data set limit)
57	KEY (no space available to add keyed record)
70	ENFILE
80	UNDEFINEDFILE (signaled)
81	UNDEFINEFILE (attribute conflict)
82	UNDEFINEDFILE (access method not supported)
83	UNDEFINEDFILE (block size not specified)
84	UNDEFINEDFILE (file cannot be opened, no DD card)
85	UNDEFINEDFILE (error initializing REGIONAL data set)
90	ENDPAGE
300	OVERFLOW
310	FIXEDOVERFLOW
320	ZERODIVIDE
330	UNDERFLOW
340	SIZE (normal)
341	SIZE (I/O)
350	STRINGRANGE
360	AREA raised in ALLOCATE statement
361	AREA raised in assignment statement
362	AREA (signaled)
500	CONDITION
510	CHECK (label)
511	CHECK (variable)
520	SUBSCRIPTRANGE
600	CONVERSION (internal) (signaled)
601	CONVERSION (I/O)
602	CONVERSION (transmit)
603	CONVERSION (error in F-format input)
604	CONVERSION (error in F-format input) (I/O)
605	CONVERSION (error in F-format input) (transmit)
606	CONVERSION (error in E-format input)
607	CONVERSION (error in E-format input) (I/O)
608	CONVERSION (error in E-format input) (transmit)
609	CONVERSION (error in B-format input)
610	CONVERSION (error in B-format input) (I/O)
611	CONVERSION (error in B-format input) (transmit)

Code	Condition/Error
612	CONVERSION (character string to arithmetic)
613	CONVERSION (character string to arithmetic) (I/O)
614	CONVERSION (character string to arithmetic) (transmit)
615	CONVERSION (character string to bit string)
616	CONVERSION (character string to bit string) (I/O)
617	CONVERSION (character string to bit string) (transmit)
618	CONVERSION (character to picture)
619	CONVERSION (character to picture) (I/O)
620	CONVERSION (character to picture) (transmit)
621	CONVERSION (P-format input - decimal)
622	CONVERSION (P-format input - decimal) (I/O)
623	CONVERSION (P-format input - decimal) (transmit)
624	CONVERSION (P-format input - character)
625	CONVERSION (P-format input - character) (I/O)
626	CONVERSION (P-format input - character) (transmit)
627	CONVERSION (P-format input - sterling)
628	CONVERSION (P-format input - sterling) (I/O)
629	CONVERSION (P-format input - sterling) (transmit)
1000	Attempt to read output file
1001	Attempt to write input file
1002	GET/PUT string length error
1003	Unacceptable output transmission error
1004	Print option on non-print file
1005	Message length for DISPLAY statements zero
1006	Illegal array datum data-directed input
1007	REWRITE not immediately preceded by READ
1008	GET STRING: unrecognizable data name
1009	Unsupported file operation
1010	File type not supported
1011	Inexplicable I/O error
1012	Outstanding READ for update exists
1013	No completed READ exists - incorrect NCP value
1014	Too many incomplete I/O operations
1015	EVENT variable already in use
1016	Implicit-OPEN failure - cannot proceed
1017	Attempt to REWRITE out of sequence
1018	ERROR condition raised if end-of-file is encountered before the delimiter while scanning list-directed or data-directed input, or if the field width in the format list of edit-directed input would take the scan beyond the end-of-file.
1019	Attempt to close file not opened in current task
1500	Short SQRT error
1501	Long SQRT error
1504	Short LOG error
1505	Long LOG error
1506	Short SIN error
1507	Long SIN error
1508	Short TAN error
1509	Long TAN error
1510	Short ARCTAN error
1511	Long ARCTAN error
1514	Short ARCTANH error
1515	Long ARCTANH error
1550	Invalid exponent in short float integer exponentiation
1551	Invalid exponent in long float integer exponentiation
1552	Invalid exponent in short float general exponentiation
1553	Invalid exponent in long float general exponentiation
1554	Invalid exponent in complex short float integer exponetiation
1555	Invalid exponent in complex long float integer exponentiation
1556	Invalid exponent in complex short float general exponentiation
1557	Invalid exponent in complex long float general exponentiation
1558	Invalid argument in short float complex ARCTAN or ARCTANH
1559	Invalid argument in long float complex ARCTAN or ARCTANH
2000	Unacceptable DELAY statement
2001	Unacceptable use of the TIME built-in function
3000	E-format conversion error
3001	F-format conversion error
3002	A-format conversion error
3003	B-format conversion error
3004	A-format input error
3005	B-format input error
3006	Picture character string error
3798	ONSOURCE or ONCHAR out of context
3799	Improper return from CONVERSION ON-unit
3800	Structure length ≥ 16**6 bytes
3801	Virtual origin of array ≥ 16**6 or ≤ -16**6
3900	Attempt to wait on inactive and incomplete event
3901	Task variable already active
3902	Event already being waited on
3903	Wait on more than 255 incomplete events
3904	Active event variable as argument to COMPLETION pseudo-variable
3905	Invalid task variable as argument to PRIORITY pseudo-variable
3906	Event variable active in assignment statement
3907	Event variable already active

Code	Condition/Error
3908	Attempt to wait on an I/O event in wrong task
8091	Invalid operation
8092	Privileged operation
8093	EXECUTE statement executed
8094	Protection violation
8095	Addressing interruption
8096	Specification interruption
8097	Data interruption
9000	Too many active ON-units and entry parameter procedures
9002	Invalid free storage (main procedure)

Return Codes and User Completion Codes

PL/I programs can terminate abnormally in six different ways:

1. EXIT (abnormal termination of a task)

2. STOP (abnormal termination of the program)

3. If the ERROR condition is raised and there is neither an ERROR on-unit nor a FINISH on-unit with a GO TO statement.

4. CALL IHEDUMP or CALL IHEDUMT.

5. If an interrupt occurs during execution of the error-handling routine. This results in execution of the ABEND macro.

6. An operating system abnormal termination in the major task.

In cases (1) to (4) above, a special return code is generated by the PL/I library and passed to the return code register of the operating system. The code generated is the sum of:

a. a basic code associated with the cause of termination, and

b. the ordinary return code. (The ordinary return code is either that set by the programmer or the default value, zero.)

The basic codes associated with the cause of termination are:

EXIT and STOP	1000
ERROR	2000
CALL IHEDUMP or IHEDUMT and when an error occurs in IHEDUMC or IHEDUMJ	3000
Error-handling routine interrupt	4000

There are two instances in which execution of the PL/I program does not commence but a return code is generated:

Pseudo-register vector too long	4004
No main procedure	4008

In cases (5) and (6) above, a full storage dump will be printed, provided that a SYSABEND or SYSUDUMP DD statement has been used. If not, and MFT or PCP is being used, an indicative dump will be printed. Case (5) results in a completion code of 4000. When a program terminates abnormally with a completion code of 4000, this means that a disastrous error has occured, such as a control block being overwritten.

When a program terminates abnormally with a completion code of 3333, this means that a disastrous error has occurred in the dump output modules. It can also mean that a program interrupt has occurred during the execution of a non-PL/I routine which does not have its own error handler and which has modified the value of register 12. (Non-PL/I routines include data management, sort, and any user-written routines invoked from a PL/I routine.)

When the operating system terminates a program, the user completion code is zero, and the system completion code is the operating system completion code.

All the 4000-series codes mentioned above are accompanied by a message at the operator's console. (For further explanation of the underlying causes of these messages, refer to the explanations given for each message in "Object-Time Error Messages" in Appendix K.

A user completion code of 4092 may result if:

1. A call to IHEREST is made to force a restart.

2. A user completion code of 4092 was not specified at system generation time as eligible for automatic restart, or no RD parameter or RD=NR was specified in the JOB statement, or if the operator replies "NO" when the system asks whether the job should restart.

See Chapter 14 and IBM System/360 Operating System: Checkpoint/Restart for further information.

Prior to Release 16 of the operating system, in the case of EXIT, STOP, and ERROR, instead of a return code an ABEND macro was issued, with a completion code equal to the return code which is now returned. The effect of the change is that now only the PL/I program is terminated when EXIT,STOP, or ERROR occur, and not the whole job step; this gives the programmer the ability to select the subsequent course of action according to the cause of termination of his PL/I program. He does so either in a program which calls the PL/I program, or in the next job step by using the COND parameter of the EXEC statement.

STEP ABEND FACILITY

Using the standard default module IHEABND in the PL/I library (data set SYS1.PL1LIB), error handling will be as stated above. The STEP ABEND facility is available however by using a user-written module IHEABND. Replacing the standard module results in a STEP ABEND being issued whenever the ERROR condition is raised or signalled in the absence of an ERROR ON-unit.

Changing Standard Module IHEABND

To change the standard module IHEABND to obtain the STEP ABEND facility the return code in register 15 must be set to non-zero. To become an installation standard this module must be link-edited into the PL/I library (data set SYS1.PL1LIB) to replace the standard module IHEABND. For use in one program only, the object deck is link-edited with that program.

The purpose of this module is to set a non-zero return code in register 15. The error handling module IHEERR calls IHEABND and examines the return code held in register 15 to decide whether STEP ABEND or normal action is required. (The standard default module IHEABND in SYS1.PL1LIB sets a zero return code in register 15). If the return code is non-zero then a STEP ABEND is issued.

RETURN CODES

Return codes are set by use of the statement CALL IHESARC, or CALL IHETSAC (multitasking).

Module form:

```
IHEABN    CSECT
          ENTRY  IHEABND       Define entry
          USING  *,15          Addressability
IHEABND   EQU    *             Entry point
          L      15, NONZERO   Set return code in register 15
          BR     14            Return  on register 14
NONZERO   DC     F'1'          Non-zero return code
          END
```

Appendix J: Implementation Conventions and Restrictions

AREA Attribute

The size of an area is limited to 32,767 bytes. In this implementation, the AREA size is provided by the value associated with the AREA attribute or by the default value of 1000 bytes.

Aggregates

The length of an aggregate is limited to 8,388,607 bytes.

Array Bounds

Arrays are limited, for each dimension, to a lower bound of -32,768 and to an upper bound of 32,767.

BASED Attribute

The implementation of offsets and pointers does not support bit addressing. This restriction has no practical effect on ALIGNED bit strings. With UNALIGNED bit strings belonging to arrays or structures, however, only offset or pointers to major structures or minor structures with byte (or higher) alignment should be used.

Based Variable Declaration

A pointer variable must be included in a based variable declaration. (This is the pointer that will be set in the absence of a SET option from a LOCATE or an ALLOCATE statement referring to the based variable.)

Based Variables

The pointer variable explicitly or implicitly qualifying a based variable must be a non-based unsubscripted scalar pointer identifier.

The BASED attribute must be followed by a pointer identifier in parentheses.

The OFFSET attribute must be followed by an identifier in parentheses.

The variable named in the OFFSET attribute must be an unsubscripted level 1 based area.

Offset variables may not be used in any SET option or in any explicit or implicit based variable qualifier.

A based variable may not have the INITIAL attribute.

A based label array cannot be initialized by means of subscripted label prefixes.

A based structure can have either:

1. One adjustable array bound, or
2. One adjustable-length bit or character string.

Based Structure with One Adjustable Array Bound: This is permitted only when there are no adjustable strings in the structure. The bound must conform to the following rules:

1. It must be of the form:

   X REFER (Y)

   where X is an unsubscripted fixed binary integer variable,

   Y is an unsubscripted fixed binary integer variable, of the same precision as X, which is:

   a. part of the structure, and

   b. not associated with an explicit pointer qualifier.

2. It must be the upper bound of the leading dimension (including inherited dimensions) of the element with which it is declared.
3. The structure member with which the bound is declared must be, or must contain, the last base element in the structure.

Example:

```
DCL 1 PARTS_LIST BASED(P),
    2 FIRM_NAME CHAR(40),
    2 REF_NO FIXED BINARY,
    2 FIRM_ADDRESS,
      3 STREET_TOWN CHAR(50),
      3 COUNTRY CHAR(30),
    2 STOCK(20:N REFER(REF_NO), 100:200),
      3 NUMBER,
        4 HERE FIXED(10,0),
        4 ORDERED,
          5 PROVISIONAL FIXED(10,0),
          5 CONFIRMED FIXED(10,0),
      3 COST FIXED(5,0);
```

Based Structure with One Adjustable-Length Bit or Character String: This is permitted only when there are no adjustable array bounds in the structure. The string must conform to the following rules:

1. It must be an element.
2. It must be the last element in the structure.
3. The length must be declared in the form:

   X REFER (Y)

   where X and Y are as described above.

Example:

```
DCL 1 TYPE_OF_HOUSE BASED(P),
    2 NUMBER_OF_FLOORS FIXED(2,0),
    2 AREA FIXED BINARY,
    2 RATES_CODE CHAR(N REFER(AREA));
```

Note: Pointer and area data types must be aligned.

Block Size

The maximum size of a block must not exceed 32,760 bytes.

See IBM System/360 Operating System: PL/I (F) Language Reference Manual for details of data aggregate size requirements necessary in calculating the block size for data sets using RECORD I/O.

Blocks in a Compilation

The number of PROCEDURE, BEGIN, and iterative DO-groups, plus the number of ON-statements, must not exceed 255.

Buffers

The number of buffers must not exceed 255.

Built-In Functions

The default value for the second argument of the FIXED built-in function is 15 for binary data, and 5 for decimal data.

The default value for the second argument of the FLOAT built-in function is 21 for binary data, and 6 for decimal data.

The length of the bit string which is the value returned by the UNSPEC built-in function is defined by the type of the argument.

Argument Type	Length of Bit String
FIXED BINARY (p,q)	16 if p < 16 32 if p > 15
FIXED DECIMAL (p,q)	8*FLOOR ((P+2)/2)
FLOAT BINARY (p)	32 if p ≤ 21 64 if p ≥ 22
FLOAT DECIMAL (p)	32 if p ≤ 6 64 if p ≥ 7
CHARACTER (n)	8*n
BIT (n)	n
POINTER	32
OFFSET	32
AREA	(Number of bytes allocated + 16)*8

The length of the string returned by the ONSOURCE and DATAFIELD built-in functions is subject to an implementation maximum of 255 characters.

Character Code

Input to the object program is assumed to be in EBCDIC mode.

CHECK Condition

If an identifier that is read in by a GET DATA statement is included in a CHECK list anywhere in the program, then the CHECK condition is raised, and will be treated as enabled unless the block containing the GET DATA statement has an explicit NOCHECK prefix. If the CHECK condition is raised, system action will be taken unless the GET DATA statement lies within the dynamic scope of an ON CHECK statement for the identifier in question.

If a READ statement with the EVENT option has a KEYTO or an INTO variable for which the CHECK condition is enabled, the value of the variable will be printed immediately after execution of the READ statement, not after the WAIT statement. Consequently, the printed values of the variable will be the old, not the new values.

The CHECK condition is not raised for the name of a procedure when:

1. The procedure that enables the CHECK condition also invokes the named procedure, and
2. The invoking procedure and the named procedure have been placed in different program segments (overlays) by the linkage editor.

CHECK Lists

The maximum number of entries in a CHECK condition, whether in a prefix list or in an ON-statement, is 510.

The maximum number of data items being checked at any point in the compilation varies between 2078-2n and 3968-2n, where n is the number of currently checked items which have the attribute EXTERNAL.

If a structure or part of a structure is in a CHECK condition, the number of items in this restriction must include all elements of the structure.

COBOL Option

If an ON-condition arises during a READ INTO, then:

1. The INTO variable may not be used in the ON-unit.
2. If the completed INTO variable is required, there must be a normal return from the ON-unit.

Collating Sequence

In the execution of PL/I programs, comparisons of character data will observe the collating sequence resulting from the representations of characters in bytes of IBM System/360 storage in extended binary coded decimal interchange code (EBCDIC).

COLUMN Format Item in Non-Print Files

For input files, if the value of the expression is greater than the current record length, a value of 1 is assumed.

For both input and output files, if the value is less than the current position on the line, the file is positioned at COLUMN(value) on the next line.

For output files, all characters from the current position in the line to the next position are blanked out. For U- or V-format records, if another record is required, a short record is put out subject to the rules described under the SKIP format item.

Concatenated Data Sets

Concatenation of data sets with 'unlike attributes' (device type, record format, etc.,) is not supported at object time.

Constants

The precision or length of constants may not be greater than the precision or length of the corresponding type of variable.

Constants, Floating-Point

The exponents of floating-point numbers are restricted to a maximum of 2 digits for decimal or 3 digits for binary.

Constants Returned by Procedures

If a procedure has more than one entry point, and each entry point returns a value, code is generated to convert each value returned to each of the data types for the entry points. If any of these values is a constant, it is possible that this constant cannot be converted to all the data types specified. A severe error message will be put out, and execution will be unsuccessful.

This situation can be avoided by assigning the constant to a variable of the same data type, and then returning this variable. For example:

```
   DCL A ENTRY RETURNS(CHAR(8)),
       B ENTRY RETURNS(FIXED DECIMAL(15)),
       C ENTRY RETURNS(BIT(64)),
       ATEMP CHAR(8);
       .
       .
       .
A: ENTRY CHAR(8);
   ATEMP='A08';
   RETURN(ATEMP);
B: ENTRY FIXED DECIMAL(15);
   RETURN(108);
C: ENTRY BIT(64);
   RETURN('10101'B);
```

The use of ATEMP avoids the interrupt caused by the CHARACTER->FIXED DECIMAL and the CHARACTER->BIT conversions. However, execution may still be unsuccessful, and a warning message is put out to remind the user.

Constants, Sterling

The maximum number of digits allowed in the pounds field of a sterling constant is 13.

The number of digits following the decimal point in the pence field must not exceed 13 minus the number of digits in the pounds field.

Constants, String

The number of characters in a string constant, after expansion of iteration factors, may not exceed the size of a dictionary block minus 14. The size of a dictionary block will vary with the storage available to the compiler in the same way as does text block size, but will not be less than 1,024 bytes.

Data-Directed Input/Output

The maximum length of a qualified name, including the separating periods, is 255 characters.

Data-Directed List

The maximum number of elements permitted in a list for data-directed input is 320. Each base element of a structure counts as a separate list element.

DECLARE Statement (compile-time)

No more than three levels of factoring are permitted in a compile-time DECLARE statement.

DEFINED Attribute

The DEFINED attribute is always evaluated on entry to the declaring block. Overlay defining is not permitted if the base is either subscripted or is declared CONTROLLED. Correspondence defining is not permitted if the base is declared CONTROLLED.

Example:

```
DCL B CONTROLLED, C(10) AUTOMATIC;

DCL A DEFINED B,       /*INVALID*/
    E DEFINED C(I),    /*INVALID*/
    F(10) DEFINED C;   /*VALID*/
```

No correspondence defining may be used with arrays of structures.

Example:

```
DCL 1 A(10), 2 B, 2 C;

DCL 1 D(5) DEFINED A, 2 E, 2 F; /*INVALID*/
DCL D(5) DEFINED B;             /*VALID*/
```

Dimensions

The maxiumum number of dimensions permitted, including dimensions inherited from containing structures, is 32.

DISPLAY Statement

The maximum lengths of character string acceptable are 72 characters for the message and 126 characters for the reply. The reply string's current length is set equal to its maximum length and padded with blanks if necessary.

Dummy Arguments

The maximum number of dummy arguments permitted at each invocation of a procedure is 64.

Edit-Directed Input/Output

In the output format item E(w,d,s), s must be less than 17 digits. In the output format item E(w,d), d must be less than 16 digits.

If the number of significant digits in E-format is greater than 16, then:

E-format input: CONVERSION condition raised

E-format output: ERROR Condition raised

When either a repetitive specification appears in an edit-directed format list or an iteration factor is used in a format list, the data items are associated with their format items during execution rather than compilation. The compiler will therefore include library modules that can handle all the possible pairings of data and format items. Some of these modules, however, will never be invoked should the circumstances that demand their use never arise.

ENTRY Names as Arguments and ON-Statements in Recursive Contexts

In the first version of the (F) compiler, ENTRY parameters were invoked with the environment existing at the time of invocation. In subsequent versions, they will be invoked with the environment existing at the time when the ENTRY name was passed as an argument.

Example:

```
P1: PROC RECURSIVE;
    B=1;
    CALL P4(P3);
    RETURN;
P4: ENTRY(PP);
    B=2;
    CALL PP;
P3: PROC;
    PUT DATA(B);
    END;
    END;
```

Note: For the first version, the above procedure gave B=2;, for subsequent versions it gives B=1;.

In the first version of the (F) compiler, ON-units in recursive contexts were entered with the environment existing when the condition occured. In subsequent versions, the ON-unit will be entered with the environment which was in existence when the ON-statement was executed.

Example:

```
P: PROC RECURSIVE;
    DCL I STATIC INIT(0),M AUTOMATIC;
    I=I+1;
    M=I;
    IF I=1 THEN DO;
    ON OVERFLOW PUT DATA(M);
    END;
    IF I=3 THEN SIGNAL OVERFLOW;
    ELSE CALL P;
    END;
```

Note: In the first version, the procedure gave M=3;, subsequent versions give M=1;.

These modifications of semantics can affect only those programs which contain both recursive procedures and either entry parameters or ON-statements.

EVENT Option

The EVENT option is implemented for RECORD input/output statements used as follows:

Access	Data-Set Organization
SEQUENTIAL UNBUFFERED	CONSECUTIVE, REGIONAL
DIRECT	CONSECUTIVE, INDEXED or REGIONAL

Note: The EVENT option should not be used on a WRITE statement if V- or U- format records are being added to a REGIONAL(3) data set which is being accessed in a direct update mode. If the EVENT option is specified in the DISPLAY statement, it must follow the REPLY option.

Exponentiation

The expression X**(-N) for N>0 is evaluated by taking the reciprocal of X**N. This may cause the OVERFLOW condition to occur as the intermediate result is computed, which corresponds to UNDERFLOW in the original expression.

Expression Evaluation

The maximum number of temporary results which may exist during the evaluation of an expression or during an assignment statement is 200.

An estimate of the number of temporary results which may exist during the evaluation of an expression can be obtained from the following:

> At each level of parenthesis, count one for each operator which is forced to be evaluated before an inner level of parentheses. For each such operator, count one for each operand which requires conversion before use, count one for each nested function, count one for each subscripted variable used as a target in a assignment statement, and finally, count one for each pseudo-variable and each argument of a pseudo-variable.

Factoring of Attributes

The total number of pairs of parentheses used for factoring attributes in DECLARE statements within a compilation is dependent upon the SIZE option, as follows:

Text Block Size	Factor Bracket Limit
1K	146
2K	292
4K	585
8K	1171
16K	2340

For relationship between SIZE option and block size, see "Control Options" in Chapter 5: Compilation.

FLOAT Attribute

Floating-point precision specified as FLOAT (*) in a parameter description within the GENERIC attribute, is not permitted.

Floating-Point Magnitude

The magnitude of a normalized floating-point variable or intermediate result must lie in the range from $2.4 * 10^{-78}$ to $7.2 * 10^{75}$.

Function Values

The maximum number of different data types or precisions returned by one function may not exceed 256.

GENERIC Attribute

There is a limitation on the number of family members and argumants which may be associated with a GENERIC entry name. the value given by evaluating the following formula must not exceed 700:

$$3n + 8\sum_{1}^{n} a_i + 8MAX(a_1, a_2, \ldots a_n) + 3d$$

where n = the number of family members.
a_i = the number of arguments relating to the ith family member.
d = the greatest function nesting depth at which an invocation of the GENERIC entry name appears.

Halfword Binary Facilities

With previous versions of the compiler, fixed binary variables of any precision were always mapped as fullwords (requiring four bytes of storage). The fifth version of the compiler will map fixed binary variables of precision less than 16 as halfwords (requiring only two bytes of storage), and will use IBM System/360 halfword instructions to process them. Note that variables of default precision will be mapped as halfwords.

The change does not apply to fixed binary constants or fixed binary intermediate targets (i.e., compiler created temporaries for holding intermediate results of operations). These will continue to occupy fullwords.

Identifiers, Length

The following types of identifiers should contain not more than seven characters:

> EXTERNAL data identifiers
> EXTERNAL PROCEDURE and ENTRY labels
> EXTERNAL Files
> CONDITION identifiers

If this restriction is exceeded, the first four characters are concatenated with the last three to form an abridged identifier.

In, addition, such identifiers must not start with the letters IHE, in case they conflict with the names of library modules.

Identifiers, Subscripted

For subscripted identifiers, the maximum number of characters in the subscript is limited to 255 characters. This figure includes the first left parenthesis, the commas, and the final right parenthesis; it excludes redundant characters such as blanks and plus signs.

INCLUDE Statement (compile-time)

Included text must be from a member of a partitioned data set. If a single identifier is specified, either as (identifier) or identifier then it is assumed to be the name of a member of the partitioned data set with the ddname SYSLIB. If $identifier_1$ ($identifier_2$) is written, then $identifier_1$ is the ddname and $identifier_2$ is the member name. DD statements must be provided for partitioned data sets used. Records in these data sets must have a fixed length of not more than 100 characters. The maximum blocking factor is 5. The source margin and character set options on the EXEC statement also apply to included text.

INDEXAREA Option

The index-area-size parameter to this option is a decimal constant and must not exceed 32,767.

INITIAL Attribute

An INITIAL attribute given for CHARACTER or BIT data of STATIC storage class may not specify 'complex expressions' as initial values.

Example:

```
DCL C CHAR(10)
    STATIC INIT(3+4I);     /*INVALID*/

DCL C CHAR(10)
    STATIC INIT('3+4I');   /*VALID*/
```

Only the INITIAL CALL form of the INITIAL attribute is allowed in pointer declarations.

INTO Option of READ Statement

When the record variable of a READ statement is a variable length bit-string, the byte count, not the bit count, is stored as the current length. This is because all variable length bit-strings are not both byte aligned and multiples of eight.

LABEL Attribute

The number of statement-label constants specified by the LABEL attribute is limited to 125 in any particular label list.

Qualified names may not be used as label prefixes. Reference to arrays of label variables initialized by means of subscripted labels must not require explicit structure qualification.

Example:

```
DCL Z(3) LABEL,
    1 S,
      2 Y(3) LABEL,
    1 S1,
      2 Y(3) LABEL,
    1 S2,
      2 X(3) LABEL;

Z(1): ;       /*VALID*/
S.Y(1): ;     /*INVALID*/
X(1): ;       /*VALID*/
```

LABEL Variables in Structures with the LIKE Attribute

Initialization of LABEL variables in these structures requires careful handling particularly as the implementation does not provide the result specified by the language. A structure A is declared, using the LIKE attribute, to be identical to a structure B. Structure B contains a LABEL variable that is initialized, using the INITIAL attribute, to the value of a LABEL constant. The initial value of the corresponding LABEL variable in A is the initial value of the LABEL constant known in the block containing the declaration of B, not A.

Example:

```
      DCL 1 B,
          2 L LABEL INIT(L1);
        .
        .
        .
 L1:    . ;     /*B.L=L1*/
        .
        .
        .
      BEGIN;
      DCL A LIKE B;
        .
 L1:    . ;     /*A.L IS GIVEN THE VALUE OF
                  L1 IN STRUCTURE B*/
        .
        .
        .
      END;
```

Label Constants in Recursive Procedures

When a label constant in a recursive procedure is the object of a GOTO statement, the branch will be taken to the label in the most recent invocation of the procedure. The most recent invocation is used irrespective of the environment that invoked the recursive procedure and of the environment that invoked the procedure containing the GOTO statement. Consequently, if the GOTO statement is in a procedure which was passed as an entry name parameter, the branch will be to the most recent invocation rather than to the invocation that was active when the entry name parameter was passed.

In the following example, the first invocation of A invokes A recursively; the second invocation of A invokes procedure B. The GOTO statement in procedure B causes a branch to the label L in the most recent invocation of procedure A rather than to the previous invocation.

```
 X: PROC OPTIONS(MAIN);
    CALL A(A);
    .
    .
 A: PROC(E) RECURSIVE;
    DCL E ENTRY,
        SW BIT(1) STATIC INIT('0'B);
    .
    .
 B: PROC
    .
    GOTO L;
    .
    END B;
    .
    .
    IF SW THEN CALL E
    SW='1'B;
    CALL A(B);
    .
    .
 L: .
    END A;
    .
    .
    END X;
```

Label Variables in Recursive Procedures

If a label variable is assigned a label constant in a particular invocation of a recursive procedure, a GOTO statement associated with the invocation will transfer control to the label constant within that invocation. Consequently, if the GOTO statement is in a procedure which was passed as an entry name parameter, the branch will be to the label constant assigned to the label variable in the environment that was active when the entry name was passed.

In the following example, the first invocation of procedure A invokes A recursively; the second invocation of A invokes procedure B. The GOTO statement in procedure B causes a transfer to the label L in the invocation of A that was active when the entry name parameter B was passed, i.e., the first invocation of A.

```
 X: PROC OPTIONS(MAIN);
    CALL A(A);
    .
    .
 A: PROC (E) RECURSIVE;
    DCL E ENTRY;
    DCL SW BIT(1) STATIC INIT('0'B);
    DCL LAB LABEL INIT(L);
    .
    .
 B: PROC;
    GOTO LAB
    END B;
    .
    .
    IF SW THEN CALL E;
    SW='1'B;
    CALL A(B);
    .
    .
 L: .
    END A;
    .
    .
    END X;
```

Level Numbers

The maximum declared level number permitted in a structure is 255. The maximum true level number permitted in a structure is 63.

Line Numbering (Compile-Time)

Where constants or comments span more than one line, the output line numbering refers to the first input line number of the string or comment.

LINESIZE Format Item

The maximum amd minimum line size depend on the record format.

Record Format	LINESIZE	
	Minimum	Maximum
V	Non-PRINT file: 10 PRINT file: 9	32,751
U,F	1	32,759

If a line size is not specified, the default values are:

PRINT file: 120 characters

Non-PRINT output file: no default value

The LINESIZE value determines the logical-record length in the data set (i.e., the value of LRECL):

F- and U-format records: LRECL = LINESIZE
V-format records: LRECL = LINESIZE + 4

For PRINT files, an extra byte (for the ANS control character) is added to each of the above LRECL values.

If BLKSIZE is specified, its value and the LRECL value must be compatible. If BLKSIZE is not specified, its value is calculated from the LINESIZE value.

Evaluation of the expression gives an integer that must be within the limits described above. For V- and U-format records, LINESIZE is the maximum size of a line. If a variable LINESIZE is required, the maximum value must be specified as the BLKSIZE in the DD statement or in the ENVIRONMENT attribute. Short lines are padded with blanks for F-format records only.

MACRO Compiler Option (compile-time)

The MACRO option should be included among the complete set of options for the compiler invocation if the program contains compile-time statements.

MAIN Option

A single parameter may be passed by the EXEC statement for the execution job step to the MAIN procedure. If this facility is used, the first parameter to the MAIN procedure should be declared as a VARYING character string; the maximum length is 100, and the current length is set equal to the parameter length at object time. The parameter can also be a fixed-length character string.

MAX,MIN,MOD, Built-In Functions

When the arguments to these functions have different attributes, all the arguments are converted, before the function is invoked, to the highest characteristics. Contrary to the language specification, both the precision and the scale factor of an argument will be adjusted. If all the arguments are FIXED, application of the highest-characteristics rule may, in conjunction with the maximum precision defined by the implementation, cause truncation and hence an inaccurate result. For example:

```
DCL X FIXED DECIMAL(12,1),
    Y FIXED DECIMAL(12,9);
Z=MOD(X,Y);
```

Here Z (whatever its attributes) will be wrong. X and Y are stored in a temporary field which would have, according to the precisions of the operands, a precision larger than the implementation permits. Therefore the implementation-defined maximum is applied, resulting in a precision of (15,9). Y can be stored satisfactorily inside such a field but X is truncated, with the loss of its five high-order digits.

When the MOD built-in function is used with FIXED arguments of different scale factors, the results may be truncated. If

SIZE is enabled, an error message will be printed; if SIZE is disabled, no error message will be printed and the result is undefined.

Multiple Assignments and Pseudo-Variables

Multiple assignments are limited by the following rule:

> Count 11 for each target of a multiple assignment, add 3 for each pseudo-variable, and then add 11 for each argument of a pseudo-variable. The total must not exceed 4,085.

Names Generated by the Compiler

The number of names generated by the compiler must not exceed 11,264 in a compilation. One name is generated for each PROCEDURE, BEGIN, or ON-block, for each variable declared as CONTROLLED INTERNAL, and for each INTERNAL file.

Names, Qualified

The number of characters in a qualified name, which is to be used either for data-directed input/output or in CHECK lists, must not exceed 256.

Note that if the DATA option without a list is used for data-directed input, this will include all structure elements in the compilation.

Nesting Limitations

There must not be more than 50 levels of nesting at any point in the compilation. The degree of nesting at any point is the number of PROCEDURE, BEGIN or DO statements, without a corresponding END statement, plus the number of currently active IF compound statements, plus the number of currently unmatched left parentheses, plus the number of dimensions in each active array expression, plus the maximum number of dimensions in each active structure expression.

The number of nested iteration factors in a format list must not exceed 20. The maximum nesting of ENTRY attributes within an ENTRY or GENERIC attribute is 3.

Nesting and Depth of Replacement

At any point in the program the combined level of nesting and depth of replacement is restricted to 50. However, since not all nested or replacement items require the same amount of space, a program may run with a greater actual nesting or replacement depth than 50 levels. Depth of replacement is self-defining, but nesting level requires some clarification. A nesting level is required for:

1. Each pair of parentheses, either explicit or implied by hierarchy of operation.
2. Each IF, DO or PROCEDURE statement.
3. Each member of a parenthesized list, such as factor lists in DECLARE statements or argument lists of procedures.

OFFSET and POINTER Built-In Functions

The OFFSET and POINTER built-in functions cannot be specified explicitly. However, if the value of an offset variable is assigned to a pointer variable, or a pointer value to an offset variable, the necessary conversion is implicit in the assignment.

ON-Units and Entry Parameter Procedures

There is an implementation limit to the number of ON-units and/or entry parameter procedures which can be active at any time. An entry parameter procedure is one that passes an entry name as parameter to a procedure it calls. The total permissible number of these ON-units and/or entry parameter procedures is 127.

ONCOUNT Built-In Function

This built-in function is supported only for Model 91 requirements.

PAGESIZE Option

The maximum size of a page is 32,767 lines; the minimum is 1 line. If the page size is not specified a value of 60 lines is assumed.

Parameters

The maximum number of parameters permitted at any entry point is 64.

PICTURE Attribute

The maximum length of a PICTURE describing a numeric field, after expansion of iteration factors, is 255.

The maximum length of a PICTURE describing a character string, after expansion of iteration factors, is the size of a dictionary block, less 14. The size of a dictionary block will vary with the storage available to the compiler in the same way as does text block size, but will not be less than 1,024 bytes (or 768 bytes if the EXTDIC option is in use).

POSITION Attribute

The maximum value of the integer constant in the POSITION attribute is 32,767.

Precision

The precision, N, for a compile-time variable declared FIXED is restricted to 5.

The maximum precision of a variable or of an intermediate result is:

53	for	FLOAT BINARY
16	for	FLOAT DECIMAL
31	for	FIXED BINARY
15	for	FIXED DECIMAL

Procedures (compile-time)

There may be no more than 254 compile-time procedures per compilation. Further, each procedure is limited to a maximum of 15 parameters.

Record Size

The maximum size of a record must not exceed 32,760 bytes. See IBM System/360 Operating System: PL/I (F) Language Reference Manual, for details of data aggregate size requirements necessary in calculating the record size for data sets using RECORD I/O.

REFER Option

The restriction on the two variables in the REFER option of the BASED attributes has been eased to permit fixed binary integer variables of the same precision as each other. This will allow the user the choice of either continuing to use fullword binary or using halfword binary for the controlling fields in self-defining structures.

Scale Factor

The scale factor of a variable, or of an intermediate result of type FIXED, must be in the range -128 and +127.

Size of Compile-Time Processor Input

The user's program is maintained internally as blocks of text. Block size is assigned at the start of processing and is a function of machine size as specified by the SIZE option on the EXEC card.The total size of internal text is restricted to 90 times the size of a text block. The minimum system configuration results in a block size of 1K, so a total of 90K is allowed for internal text. This minimum figure is roughly equivalent to 1000 source input statements.

Size of Individual Statement

All statements, other than a DECLARE statement, are limited to 3,500 source characters, i.e., equivalent to 50 cards. The 'content' of any statement, other than a DECLARE statement, is limited by the size of a text block; this varies, as described in the preceding paragraphs, with the storage available, but will not be less than 1,024 bytes.

The contect of a statement can be calculated by ignoring nonsignificant blanks and comments, expanding iteration factors in string constants and pictures, and then adding one byte for each occurrence of an identifier, and three bytes for each occurrence of a constant.

To this, for binary constants add the iterations of any CHARACTER or BIT strings (note that at this point BIT strings are treated as characters, not bits), since the (F) compiler expands the strings as if the programmer had written then in full, and two decimal digits for decimal constants. At most, these restrictions will limit a statement to six cards, but the limit will normally be between 20 and 30 cards, even for a text block of 1,024 bytes.

These restrictions also apply to a DECLARE statement for the text between any two commas which are not contained within parentheses.

SKIP Format Item in Non-Print Files

For output files, SKIP action depends on the record format:

F-format: On a short line, SKIP fills out the remainder of the line with blanks.

V-format: SKIP puts out the current line as a short record. If the byte count of the line is less than 14 (18 with control bytes), the line will be blanked up to that size. Successive lines will be of the same minimum length, padded with blanks.

U-format: SKIP will put out the current line as a short record.

Statements

The total size of the internal text, at any point in the compilation, is restricted to 90 times the size of a text block. The size of a text block is itself dependent on the amount of main storage available to the compiler, as specified by the SIZE option. The minimum block size is 1,024 bytes (1K), giving a maximum size for the internal text of 92,260 (90K). This is equivalent to roughly 280 executable statements.

The maximum block size is 16,382 bytes, giving a maximum of 1,474,560 bytes for the size of internal text. This is equivalent to roughly 14,000 executable statements.

The figures given for numbers of statements are necessarily approximate, since the number of bytes per statement will vary between different types of source programs.

STRING Built-In Function

The argument may be an element array, or structure variable that consists of one of the following:

1. Bit strings
2. Character strings
3. Decimal numeric pictures
4. A mixture of (2) and (3)

It cannot be an operational expression.

The argument can be ALIGNED or UNALIGNED; if it is ALIGNED, padding is not included in the result.

The concatenated string in the result has a maximum length of 32,767 bytes.

String Lengths

The length, in characters or bits, of a string variable or intermediate string result is limited to 32,767.

String Lengths in Intermediate Result Fields

When non-adjustable VARYING strings, or functions which return non-adjustable VARYING strings, are used in an expression, the lengths of the intermediate result fields are calculated from the maximum lengths of the operands. If these lengths are at or near the maximum permitted by the implementation (32767 bytes or bits), the length of the intermediate fields may be greater than the implementation maximum; if so, they will be truncated on the left. This situation can occur with concatenation, the UNSPEC function with a character-string argument, the REPEAT function, and the STRING function.

The use of adjustable VARYING strings can create a similar problem. When an operand of the concatenate operator or the argument of the UNSPEC function is an adjustable VARYING string, the length of the intermediate result field is not tested, and execution will fail. This situation can also occur with SUBSTR if the third argument is not a constant, because in this case the result is an adjustable VARYING string.

Similarly, when a VARYING string is passed as an argument to a fixed-length string parameter, the length of the temporary argument created is the maximum length. If the user wishes to pass the current length of the VARYING string (in, for example, Y=X(A)), a possible method is:

```
DCL ATEMP CHAR(*) CTL;
ALLOCATE ATEMP CHAR(LENGTH(A));
ATEMP=A;
Y=X(ATEMP);
FREE ATEMP;
```

Structure and Array Expressions

The level of nesting in structure and array expressions is limited by the following rule:

> For each level of nesting of structure or array expressions, add 2 for the maximum number of dimensions in the structure or array, add 2 for the maximum level in a structure expression, add 3 for each subscript or argument list in the expression or assignment, and finally, add 15.

The total for the whole nest should not exceed 900.

Structures

No references can be made to cross sections of arrays of structures; the whole of an array of structures may be referenced, or a single element of the array may be referenced, but not a cross section.

Example:

```
DCL 1 A(10,10), 2 B, 2 C(10,10);
DCL 1 X(10), 2 Y, 2 Z(10,10);

A(*,J)=X;       /*INVALID*/
A=A+1;          /*VALID*/
A(I,J)=X(I);    /*VALID*/
```

A cross-section of an EVENT array is not permitted to appear in a WAIT statement.

Example:

```
DCL EVT(10,10,2) EVENT;
WAIT(EVT) 200;          /*VALID*/
WAIT(EVT(I,J,2)) 100;   /*VALID*/
WAIT(EVT(1,*,1)) 10;    /*INVALID*/
```

No structure or array of structures may be passed as an argument to either a built-in function (except STRING, ADDR, and ALLOCATION), or to a procedure declared with the attribute GENERIC.

Example:

```
DCL 1 A(10), 2 B;

A=SIN(A);    /*INVALID*/
B=SIN(B);    /*VALID*/
```

No reference may be made to both a structure and an array of structures in the same expression or assignment.

Example:

```
DCL 1 A(10), 2 B, 2 C,
    1 F(10), 2 B, 2 C,
    1 P, 2 Q, 2 R;

A=P;                /*INVALID*/
A=F;                /*VALID*/
A(I)=P;             /*VALID*/
A=F, BY NAME;       /*VALID*/
A=F(I), BY NAME;    /*INVALID*/
```

TITLE Option

If the TITLE option specified exceeds eight characters, then the first eight are used. If there is no TITLE option, the file name (padded or truncated to eight characters) is used as the ddname.

TRANSIENT Attribute

The following rules apply specifically to the use of TRANSIENT with the (F) compiler:

1. The TRANSIENT attribute can be specified only for RECORD KEYED BUFFERED files with either the INPUT or the OUTPUT attribute.
2. The ENVIRONMENT attribute with one of the two teleprocessing format options (G and R) must be declared for TRANSIENT files.
3. Input can be specified only by a READ statement with the KEYTO option and either the INTO option or the SET option.
4. Output can be specified only by a WRITE statement or a LOCATE statement, either of which must have the KEYFROM option.
5. The EVENT option is not permitted since TRANSIENT files are always BUFFERED.

6. The 'data set' associated with a TRANSIENT file is in fact a queue of messages maintained automatically in main storage by a separate message control program using the QTAM (Queued Telecommunications Access Method) facilities of the operating system. The queue is always accessed sequentially.

7. The name or title of a TRANSIENT INPUT file must be the name of a recognized queue set up by the message control program. For TRANSIENT OUTPUT files, any name can be declared, since the file is reassociated for each output operation with a queue determined by the terminal name.

8. The element expression specified in the KEYFROM option must have as its value a recognized terminal or process queue identification.

Variables

The maximum number of variables in the source program depends on the total size of the dictionary, which (for NOEXTDIC) is restricted to approximately 65,000 bytes. This is equivalent to a restriction of roughly 1,200 variables for a scientific user and to 1,000 for a commercial user. In computing these figureas a reasonable allowance has been made for constants, statement labels, and other items which may require dictionary entries.

If the EXTDIC option is specified, the maximum size of the dictionary is approximately 1.5 times 65,000 bytes for a block size of 1K, and approximately 3.5 times 65,000 bytes for other block sizes.

The figures for variables are necessarily approximate, since the size of a dictionary entry varies with the type of variable, length of identifier, whether it is a structure element, and so on.

Variables at Compile-Time

The maximum number of compile-time variables which can be used in a program depends on the total size of the dictionary, which (for NOEXTDIC) is restricted to 65,000 bytes. Assuming an average dictionary entry size of 28 bytes, this restricts the processor to approximately 2,300 items. An entry is made in the dictionary for each macro variable, macro procedure name, INCLUDE identifier, macro label, and unique compile-time constant. In addition, two dictionary entries are created for each iterative DO, one for each THEN or ELSE clause, and one for each compile-time procedure. Error message references are also entered into the dictionary. The dictionary is cleared at the end of compile-time processing; it is therefore unnecessary to keep the above considerations in mind if estimating available dictionary space during actual program compilation. Under EXTDIC the number of variables can be increased by a factor of 1.5 for 1K blocks, or 3.5 for larger block sizes.

VARYING Attribute

The only form of varying strings permitted in the INTO or FROM options in RECORD I/O are unsubscripted level 1 varying strings that are not members of arrays or structures.

Varying Strings passed as Arguments

If a structure passed as an argument includes a varying string, the length of the string cannot be changed within the invoked procedure unless a dummy argument is created.

Example:

```
DCL 1 P(10),
      2 Q(10),
        3 R FIXED(5,0),
        3 S CHAR(5) VAR,
      2 T FIXED(5,0);

CALL PROC1(P.Q(4,4));  /*INVALID*/
CALL PROC1(P.Q,4,4);   /*VALID; ARRAY OF
                         STRUCTURES PASSED
                         WITH REQUIRED
                         SUBSCRIPTS*/
```

WAIT Statement

If the user wishes to specify more than one event name in a WAIT statement, the multiple-wait option must have been specified at SYSGEN time.

If a WAIT statement is executed and the events required to satisfy the WAIT contain a mixture of I/O and non-I/O events all non-I/O events will be set complete before any of the I/O events.

48-Character Set

48-character set 'reserved' words (e.g., GET, LE, CAT, etc.,) must be preceded and followed by a blank or a comment. If they are not, the interpretation by the compiler is undefined and may not, therefore, be what the user intended.

A record containing part or all of a 48-character-set reserved word must be 3 characters or more in length.

Appendix K: Diagnostic Messages

Source Program Diagnostic Messages

All source program diagnostic messages produced are written in a group following the source program listing and any other listings specified as a parameter on the EXEC statement card.

Each message number is of the form IEMnnnnI, where the code IEM indicates the PL/I (F) compiler, and nnnn the number of the message. The letter I is a system standard action code indicating an informative message for the programmer.

There are four types of diagnostic message: warning, error, severe error, and termination error.

A Warning is a message that calls attention to a possible error, although the statement to which it refers is syntactically valid. In addition to alerting the programmer, it may assist him in writing more efficient programs in the future.

An Error message describes an attempt to correct an erroneous statement; the programmer is informed of the correction. Errors do not normally terminate processing of the text.

A Severe error message indicates an error which cannot be corrected by the compiler. The incorrect section of the program is deleted, but compilation is continued. Where reasonable, the ERROR condition will be raised at object time, if execution of an incorrect source statement is attempted.

A Termination error message describes an error which, when discovered, forces the termination of the compilation.

The choice of the severity level at and above which diagnostic messages appear on the output is an option which may be selected by the programmer. FLAGW is assumed if no level is specified.

In the list of diagnostic messages below, the abbreviations W, E, S, and T, respectively, are used to indicate the severity of the message, and appear immediately before the number of the message. They do not appear in this way in the compiler output listings; instead, the messages are printed in separate groups according to severity.

In the following text, messages are followed where necessary by an explanation, a description of the action taken by the system, and the response required from the user. "Explanation" and "System Action" are given only when this information is not contained in the text of the message. When no "Programmer Response" is stated explicitly, the programmer should assume that he must correct the error in his source program unless the action taken by the system makes it unnecessary for him to do so. However, even when system action successfully corrects an error, the programmer should remember that if he subsequently recompiles the same program, he will get the same diagnostic message again unless he has corrected the source error.

If a problem arises in using the PL/I (F) Compiler, do the following before calling IBM for programming support:

- If the compiler preprocessor has been used and the problem occurs in the processor stage, use the 'MACDK' option to obtain the macro deck output; this ensures that source code from INCLUDE libraries is available. Otherwise, retain the source input to the compiler. (Note: if the problem occurs in the preprocessor stage, ensure that all source data in libraries as well as the primary input is available.)
- Obtain listings of SYS1.LINKLIB and of SYS1.PL1LIB, and listings of program temporary fixes (PTF's) using IMAPTFLS against these libraries.
- Specify MSGLEVEL=(1,1) on the job statement.
- Include a SYSUDUMP DD statement for the failing job step.
- If the problem is associated with a compiler termination message, recompile with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler.

Note: It may be necessary to override the default queue space for SYSOUT for the SYSPRINT data set in order to prevent a 'B37' abend.

```
e.g.  //PL1L.SYSPRINT DD SYSOUT=A,
            SPACE=(629,(2000,20))
```

E IEM0002I INVALID PREFIX OPERATOR IN STATEMENT NUMBER xxx. REPLACED BY PLUS.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0003I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0004I OPERATOR .NOT. IN STATEMENT NUMBER xxx USED AS AN INFIX OPERATOR. IT HAS BEEN REPLACED BY .NE.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0005I RIGHT PARENTHESIS INSERTED AFTER SINGLE PARENTHESIZED EXPRESSION IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0006I RIGHT PARENTHESIS INSERTED AT END OF SUBSCRIPT, ARGUMENT OR CHECK LIST IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0007I IDENTIFIER MISSING IN STATEMENT NUMBER xxx . A DUMMY IDENTIFIER HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0008I RIGHT PARENTHESIS INSERTED AT END OF CALL ARGUMENT LIST OR OTHER EXPRESSION LIST IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0009I A LETTER IMMEDIATELY FOLLOWS CONSTANT IN STATEMENT NUMBER xxx. AN INTERVENING BLANK IS ASSUMED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0010I IMPLEMENTATION RESTRICTION. IDENTIFIER yyyy IN OR NEAR STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN SHORTENED.

Explanation: Implementation restriction. Identifiers may not exceed 31 characters in length.

System Action: Identifier has been shortened by concatenating first 16 characters with last 15.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0011I CONSTANT IMMEDIATELY FOLLOWS IDENTIFIER IN STATEMENT NUMBER xxx. AN INTERVENING BLANK IS ASSUMED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0012I EXPONENT MISSING IN FLOATING-POINT CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx . ZERO HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0013I INTEGER yyyy TOO LONG IN STATEMENT NUMBER xxx . IT HAS BEEN TRUNCATED ON THE RIGHT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0014I EXPONENT TOO LONG IN FLOATING-POINT CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx . IT HAS BEEN TRUNCATED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0015I SOLITARY DECIMAL POINT FOUND IN OPERAND POSITION IN STATEMENT NUMBER xxx . A FIXED-POINT ZERO HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0016I FLOATING-POINT CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN TRUNCATED ON THE RIGHT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0017I ZERO INSERTED IN FLOATING-POINT CONSTANT BEGINNING .E IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0018I ZERO INSERTED IN PENCE FIELD OF STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0019I POUNDS FIELD IN STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN TRUNCATED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0020I ZERO INSERTED IN POUNDS FIELD OF STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0021I DECIMAL POINT IN EXPONENT FIELD OF CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx . FIELD TRUNCATED AT DECIMAL POINT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0022I DECIMAL PENCE TRUNCATED IN STERLING CONSTANT BEGINNING yyyy STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0023I LETTER L MISSING FROM STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

System Action: None

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0024I SHILLINGS FIELD TRUNCATED IN STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0025I ZERO INSERTED IN SHILLINGS FIELD OF STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0026I ILLEGAL CHARACTER IN APPARENT BIT STRING yyyy IN STATEMENT NUMBER xxx . STRING TREATED AS A CHARACTER STRING.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0027I FIXED-POINT CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx HAS BEEN TRUNCATED ON THE RIGHT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0028I LABEL REFERENCED ON END STATEMENT NUMBER xxx CANNOT BE FOUND. END TREATED AS HAVING NO OPERAND.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0029I INVALID CHARACTER IN BINARY CONSTANT IN STATEMENT NUMBER

xxx . CONSTANT TREATED AS DECIMAL CONSTANT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0030I POINTER QUALIFIER FOLLOWS EITHER A SUBSCRIPT OR ANOTHER POINTER QUALIFIER IN STATEMENT NUMBER xxx.

System Action: As stated in a further message referring to the same statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0031I OPERAND MISSING IN OR FOLLOWING STATEMENT NUMBER xxx . DUMMY OPERAND INSERTED.

Explanation: Something invalid has been found in an expression, or where an expression was expected but not found. In order that further diagnosis can be made, the compiler has inserted a dummy operand. This may cause further error messages to appear for this statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0033I AN INVALID PICTURE CHARACTER IMMEDIATELY FOLLOWS TEXT yyyy IN STATEMENT NUMBER xxx. THE PICTURE HAS BEEN TRUNCATED AT THIS POINT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0034I A LETTER IMMEDIATELY FOLLOWS A CONSTANT AT nnnn SEPARATE POSITION(S) IN STATEMENT NUMBER xxx. AN INTERVENING BLANK HAS BEEN ASSUMED IN EACH CASE.

Programmer Response: Probable user error. Check that the system action will have the required effect. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0035I LETTER F IS NOT FOLLOWED BY LEFT PARENTHESIS IN PICTURE IN STATEMENT NUMBER xxx. ONE HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0037I ZERO INSERTED IN SCALING FACTOR IN PICTURE yyyy IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0038I RIGHT PARENTHESIS INSERTED AFTER SCALING OR REPLICATION FACTOR IN PICTURE yyyy IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0039I NO CHARACTER FOLLOWS REPLICATION FACTOR IN PICTURE yyyy IN STATEMENT NUMBER xxx. THE PICTURE HAS BEEN TRUNCATED AT THE LEFT PARENTHESIS OF THE REPLICATION FACTOR.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0040I A REPLICATION FACTOR OF 1 HAS BEEN INSERTED IN PICTURE yyyy IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0043I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

Explanation: Right parenthesis missing from length attached to character or bit string.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0044I IN STATEMENT NUMBER xxx PRECISION NOT AN INTEGER

Explanation: Precision should be an unsigned integer

System Action: The action taken depends on whether the precision is found in a DECLARE statement or a PROCEDURE statement. A further message will be produced.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0045I ZERO INSERTED IN FIXED PRECISION SPECIFICATION IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0046I RIGHT PARENTHESIS INSERTED AFTER PRECISION SPECIFICATION IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0048I RIGHT PARENTHESIS INSERTED IN FILE NAME LIST IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0049I THE COMMENT FOLLOWING THE LOGICAL END OF PROGRAM HAS NOT BEEN TERMINATED.

Explanation: A /* was found following the logical end of the program and was interpreted as the start of a comment, but end-of-file was reached before the comment was terminated.

System Action: All text following the /* is read as a comment.

Programmer Response: Probable user error. Check if this is a delimiter in the wrong column of the record. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0050I INVALID STATEMENT LABEL CONSTANT IN LABEL ATTRIBUTE IN STATEMENT NUMBER xxx. THE STATEMENT LABEL CONSTANT LIST HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0051I MISSING RIGHT PARENTHESIS INSERTED FOLLOWING STATEMENT LABEL CONSTANT IN LABEL ATTRIBUTE IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0052I INVALID ATTRIBUTE IN RETURNS ATTRIBUTE LIST IN STATEMENT NUMBER xxx. THE INVALID ATTRIBUTE HAS BEEN DELETED FROM THE LIST.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0053I SURPLUS COMMA HAS BEEN FOUND IN DECLARE OR ALLOCATE STATEMENT NUMBER xxx. THIS COMMA HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0054I ILLEGAL FORM OF CALL STATEMENT. STATEMENT NUMBER xxx DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0055I LABEL OR LABELS ON DECLARE STATEMENT NUMBER xxx HAVE BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0056I NULL PICTURE FORMAT ITEM IN STATEMENT NUMBER xxx. THE CHARACTER 9 HAS BEEN INSERTED IN THE PICTURE.

Explanation: The null picture may be the result of the compiler truncating an invalid picture.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0057I INVALID CHARACTER FOLLOWING ITERATION FACTOR IN PICTURE BEGINNING yyyy IN STATEMENT NUMBER xxx. THE PICTURE HAS BEEN TRUNCATED AT THE LEFT PARENTHESIS OF THE ITERATION FACTOR.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0058I ITERATION FACTOR IN PICTURE BEGINNING yyyy NOT AN UNSIGNED INTEGER IN STATEMENT NUMBER xxx. THE PICTURE HAS BEEN TRUNCATED AT THE LEFT

PARENTHESIS OF THE ITERATION FACTOR.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0059I MISSING RIGHT PARENTHESIS INSERTED IN POSITION ATTRIBUTE IN STATEMENT NUMBER xxx.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0060I POSITION MISSING IN POSITION ATTRIBUTE IN STATEMENT NUMBER xxx. POSITION OF 1 INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0061I MISSING LEFT PARENTHESIS INSERTED IN POSITION ATTRIBUTE IN STATEMENT NUMBER xxx.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0062I THE ATTRIBUTE 'PACKED' IN DECLARATION STATEMENT NUMBER xxx IS NOW OBSOLETE, AND HAS BEEN IGNORED.

Explanation: PACKED has been removed from the language; the complementary attribute to ALIGNED is now UNALIGNED.

System Action: Since PACKED applied only to arrays and major structures, the new alignment defaults will be compatible with those of earlier versions of the compiler, except for bit string arrays that are not members of structures.

Programmer Response: Probable user error. Correct source, and recompile if necessary. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0063I MISSING LEFT PARENTHESIS INSERTED IN RETURNS STATEMENT NUMBER xxx.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0064I ILLEGAL STATEMENT FOLLOWS THE THEN IN STATEMENT NUMBER xxx. SEMICOLON HAS BEEN INSERTED AFTER THE THEN.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0067I EQUAL SYMBOL HAS BEEN INSERTED IN DO STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0069I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM CONTAINS TOO MANY BLOCKS.

System Action: Compilation is terminated

Programmer Response: Probable user error. Rewrite program

with fewer blocks, or divide into more than one separate compilation. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM0070I BEGIN STATEMENT NUMBER xxx IS NESTED BEYOND THE PERMITTED LEVEL. COMPILATION TERMINATED.

Programmer Response: Probable user error. Reduce level of nesting of blocks to 50 or less. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM0071I TOO MANY PROCEDURE, BEGIN, ITERATIVE DO, ON STATEMENTS IN THIS PROGRAM. COMPILATION TERMINATED.

Explanation: There is an implementation restriction on the number of blocks in a compilation. Refer to Appendix J of this publication for details.

Programmer Response: Probable user error. Subdivide program into two or more compilations. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0072I DO STATEMENT NUMBER xxx REPLACED BY BEGIN STATEMENT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0074I THEN INSERTED IN IF STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0075I NO STATEMENT FOLLOWS THEN IN IF STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0076I NO STATEMENT FOLLOWS ELSE IN OR FOLLOWING STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0077I ELSE DELETED IN OR FOLLOWING STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0078I IMPLEMENTATION RESTRICTION. TOO MANY CHARACTERS IN INITIAL LABEL ON STATEMENT NUMBER xxx. LABEL IGNORED.

Explanation: There is an implementation restriction on the number of characters in the subscript of a subscripted identifier. The maximum permissible number is 225.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0080I EQUAL SYMBOL HAS BEEN INSERTED IN ASSIGNMENT STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0081I LABELS OR PREFIX OPTIONS BEFORE ELSE TRANSFERRED TO STATEMENT NUMBER xxx

Explanation: Labels or prefix options illegal before ELSE and therefore transferred to following statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0082I OPERAND MISSING IN CHECK LIST IN OR FOLLOWING STATEMENT NUMBER xxx. DUMMY INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0083I ON-CONDITION INVALID OR MISSING IN STATEMENT NUMBER xxx. ON ERROR HAS BEEN ASSUMED.

System Action: ON ERROR inserted in place of invalid condition

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0084I THE I/O ON-CONDITION IN STATEMENT NUMBER xxx HAS NO FILENAME FOLLOWING IT. SYSIN IS ASSUMED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0085I COLON MISSING AFTER PREFIX OPTION IN OR FOLLOWING STATEMENT NUMBER xxx. ONE HAS BEEN ASSUMED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0090I THERE ARE NO COMPLETE STATEMENTS IN THIS PROGRAM. COMPILATION TERMINATED.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

W IEM0094I RECORD IN OR FOLLOWING STATEMENT NUMBER xxx IS SHORTER THAN THE SPECIFIED SOURCE START. THE OUTPUT RECORD HAS BEEN MARKED WITH AN ASTERISK AND IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0095I LABEL ON STATEMENT NUMBER xxx HAS NO COLON. ONE IS ASSUMED.

Explanation: The compiler has encountered an identifier which appears to be a statement label, but without a colon.

System Action: A colon is inserted

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0096I SEMI-COLON NOT FOUND WHEN EXPECTED IN STATEMENT NUMBER xxx . ONE HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0097I INVALID CHARACTER HAS BEEN REPLACED BY BLANK IN OR FOLLOWING STATEMENT NUMBER xxx. THE CONTAINING OUTPUT RECORD IS MARKED BY AN ASTERISK.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0099I LOGICAL END OF PROGRAM OCCURS AT STATEMENT NUMBER xxx. THIS STATEMENT HAS BEEN IGNORED SO THAT SUBSEQUENT STATEMENTS MAY BE PROCESSED.

Explanation: Although the compiler has detected the end of the program, there is more text following it. The programmer appears to have made an error in matching END statements with PROCEDURE, BEGIN, DO or ON statements.

System Action: The END statement is ignored

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0100I END OF FILE FOUND IN OR AFTER STATEMENT NUMBER xxx, BEFORE THE LOGICAL END OF PROGRAM.

System Action: If the statement is incomplete, it is deleted. Whether or not the statement is incomplete, the required number of END statements are added to the program so that compilation can continue.

Programmer Response: Probable user error. Correct the source code. Possible causes of this error include:

1. Unmatched quote marks
2. Insufficient END statements
3. Omission of final semicolon.

If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0101I PARAMETER MISSING IN STATEMENT NUMBER xxx . A DUMMY HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0102I LABEL MISSING FROM PROCEDURE STATEMENT NUMBER xxx. A DUMMY LABEL HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0103I LABEL MISSING FROM ENTRY STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0104I ILLEGAL STATEMENT FOLLOWS ELSE IN STATEMENT NUMBER xxx

System Action: Null statement inserted

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0105I ILLEGAL STATEMENT FOLLOWS ON IN STATEMENT NUMBER xxx

System Action: Null statement inserted

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0106I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM CONTAINS TOO MANY BLOCKS.

System Action: Compilation is terminated

Programmer Response: Probable user error. Rewrite program with fewer blocks, or divide into more than one separate compilation. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM0107I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG. THIS STATEMENT MAY CONTAIN UNMATCHED QUOTE MARKS.

Programmer Response: Probable user error. Subdivide statement and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0108I ENTRY STATEMENT NUMBER xxx IN AN ITERATIVE DO GROUP HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0109I TEXT BEGINNING yyyy IN OR FOLLOWING STATEMENT NUMBER xxx HAS BEEN DELETED.

Explanation: The source error is detailed in another message referring to the same statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0111I FIRST STATEMENT NOT A PROCEDURE STATEMENT. A DUMMY PROCEDURE STATEMENT HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0112I ENTRY STATEMENT NUMBER xxx IN BEGIN BLOCK HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0113I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

Explanation: Parenthesized list in ON statement is either not closed or contains an error and has been truncated.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0114I RIGHT PARENTHESIS INSERTED IN PREFIX OPTION IN OR FOLLOWING STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0115I LEFT PARENTHESIS INSERTED AFTER WHILE IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0116I PREFIX OPTION FOLLOWS LABEL IN STATEMENT NUMBER xxx. PREFIX OPTION IS IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0117I DATA ATTRIBUTE LIST IN PROCEDURE OR ENTRY STATEMENT NUMBER xxx IS NOT PRECEDED BY RETURNS ATTRIBUTE AND IS NOT PARENTHESIZED. RETURNS AND PARENTHESES HAVE BEEN ASSUMED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0118I OFFSET ATTRIBUTE NOT FOLLOWED BY PARENTHESIZED BASED VARIABLE IN STATEMENT NUMBER xxx. THE ATTRIBUTE IS IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0119I THE RETURNS ATTRIBUTE IN PROCEDURE OR ENTRY STATEMENT NUMBER xxx IS NOT FOLLOWED BY A PARENTHESIZED DATA ATTRIBUTE LIST. RETURNS HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0120I DATA ATTRIBUTE LIST FOLLOWING RETURNS IN PROCEDURE OR ENTRY STATEMENT NUMBER xxx IS NOT PARENTHESIZED. PARENTHESES HAVE BEEN ASSUMED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0121I THE ATTRIBUTE USES OR SETS IN STATEMENT NUMBER xxx IS OBSOLETE AND HAS BEEN IGNORED TOGETHER WITH ITS PARENTHESIZED ITEM LIST.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0122I THE ATTRIBUTE NORMAL OR ABNORMAL IN STATEMENT NUMBER xxx IS OBSOLETE AND HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0123I THE DATA ATTRIBUTE LIST IN PROCEDURE OR ENTRY STATEMENT NUMBER xxx HAS NO CLOSING PARENTHESIS. ONE HAS BEEN ASSUMED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0124I INVALID ATTRIBUTE IN DECLARE OR ALLOCATE STATEMENT NUMBER xxx. ATTRIBUTE TEXT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0125I INVALID USE OF LABEL yyy ON ON-UNIT BEGINNING AT STATEMENT xxx. LABEL HAS BEEN DELETED.

Explanation: An on-unit cannot be referenced by a label.

System Action: The label is ignored.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0126I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx HAS TOO MANY ERRORS TO BE INTERPRETED. THE STATEMENT HAS BEEN DELETED.

Programmer Response: Probable user errors. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0127I INVALID TEXT IN PREFIX OPTIONS LIST. THE TEXT BEGINNING yyy TO THE END OF THE OPTIONS LIST HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0128I LENGTH OF BIT OR CHARACTER STRING MISSING IN STATEMENT NUMBER xxx. LENGTH 1 INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0129I INVALID WAIT STATEMENT NUMBER xxx DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0130I OPERAND MISSING. COMMA DELETED IN WAIT STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0131I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0132I DUMMY OPERAND INSERTED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0134I IMPLEMENTATION RESTRICTION. TOO MANY LEVELS OF REPLICATION IN INITIAL ATTRIBUTE IN STATEMENT NUMBER xxx. THE ATTRIBUTE HAS BEEN DELETED.

Explanation: The implementation restriction on levels of nesting has been contravened. For details, refer to Appendix J of this publication.

Programmer Response: Probable user error. Rewrite INITIAL attribute with lower level of replication. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0135I AN EXPRESSION APPEARS ILLEGALLY ON THE LEFT HAND SIDE OF AN ASSIGNMENT STATEMENT. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0136I 'IN' CLAUSE IN STATEMENT NUMBER xxx HAS NO ASSOCIATED 'SET' CLAUSE.

Explanation: An IN clause must be accompanied by a SET clause in the same statement.

System Action: The IN clause is ignored

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0138I SOLITARY I FOUND WHERE A CONSTANT IS EXPECTED IN INITIAL ATTRIBUTE IN STATEMENT NUMBER xxx. FIXED DECIMAL IMAGINARY 1I HAS BEEN ASSUMED.

Explanation: The programmer has initialized an element using the variable I where the constant 1I was expected.

System Action: 1I is assumed

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0139I TEXT IMMEDIATELY FOLLOWING yyyy IN INITIAL ATTRIBUTE IS ILLEGAL. INITIAL ATTRIBUTE DELETED IN STATEMENT NUMBER xxx

Explanation: A language feature has been used that is not supported by this version of the compiler. For details, refer to Appendix J of this publication. Although the message states that the error follows the quoted text, the quoted text may itself be invalid, and the compiler may have attempted to correct the source error. In this case, there will usually be another diagnostic message associated with the statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0140I NO IDENTIFIER FOUND IN DECLARE STATEMENT NUMBER xxx. STATEMENT REPLACED BY NULL STATEMENT.

Explanation: Either no identifiers appear in the DECLARE statement or, as a result of previous compiler action, all identifiers have been deleted from the statement.

System Action: Null statement assumed.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0144I RETURNS ATTRIBUTE IS NOT FOLLOWED BY A DATA DESCRIPTION IN STATEMENT NUMBER xxx. THE RETURNS ATTRIBUTE HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0145I DUMMY IDENTIFIER INSERTED IN GENERIC ATTRIBUTE LIST IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0147I THE USE OF REFER IN STATEMENT NUMBER xxx IS EITHER INVALID OR IS NOT IMPLEMENTED IN THIS RELEASE

Explanation: The implementation of the REFER option is restricted; see Appendix J, 'Implementation Conventions and Restrictions'.

System Action: Ignore the REFER clause. A further message identifying the invalid text will usually accompany this message.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0148I LEFT PARENTHESIS MISSING IN STATEMENT NUMBER xxx

System Action: See further messages relating to this statement

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0149I COMMA HAS BEEN DELETED FROM LIST IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0150I STATEMENT NUMBER xxx IS AN INVALID FREE STATEMENT. THE STATEMENT HAS BEEN DELETED.

Explanation: The format of the statement is invalid

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0151I SEMI-COLON INSERTED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0152I TEXT BEGINNING yyyy IN STATEMENT NUMBER xxx HAS BEEN DELETED.

Explanation: The source error may be detailed in another message referring to the same statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0153I THE ATTRIBUTED BASED HAS BEEN ASSUMED IN STATEMENT NUMBER xxx WHERE CONTROLLED WAS SPECIFIED.

Explanation: The PL/I feature CONTROLLED (pointer) has been changed to BASED (pointer).

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0154I IMPLEMENTATION RESTRICTION IN STATEMENT NUMBER xxx. BASED MUST BE FOLLOWED BY AN IDENTIFIER IN PARENTHESIS.

System Action: Text is deleted. See further error message for this statement.

Programmer Response: Probable user error. Correct source statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0158I ZERO STRUCTURE LEVEL NUMBER DELETED IN DECLARE STATEMENT NUMBER xxx

Explanation: Zero level number not allowed

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0159I SIGN DELETED PRECEDING STRUCTURE LEVEL NUMBER IN DECLARE STATEMENT NUMBER xxx

Explanation: The level number must be an unsigned integer

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0163I FORMAT LIST MISSING, (A) INSERTED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0166I OPERAND MISSING IN GO TO STATEMENT NUMBER xxx. DUMMY IS INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0172I LEFT PARENTHESIS INSERTED IN DELAY STATEMENT NUMBER xxx

Explanation: The expression in a DELAY statement should be contained in parentheses

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0180I EQUAL SYMBOL HAS BEEN INSERTED IN DO SPECIFICATIONS IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0181I SEMICOLON INSERTED IN STATEMENT NUMBER xxx

Explanation: An error has been discovered. A semi-colon is therefore inserted and the rest of the statement is skipped.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0182I TEXT BEGINNING yyyy SKIPPED IN OR FOLLOWING STATEMENT NUMBER xxx

Explanation: The source error is detailed in another message referring to the same statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0185I OPTION IN GET/PUT STATEMENT NUMBER xxx IS INVALID AND HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0187I DATA LIST MISSING IN STATEMENT NUMBER xxx. OPTION DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0191I DUMMY OPERAND INSERTED IN DATA LIST IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0193I RIGHT PARENTHESIS INSERTED IN DATA LIST IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0194I MISSING RIGHT PARENTHESIS INSERTED IN FORMAT LIST IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0195I INVALID FORMAT LIST DELETED IN STATEMENT NUMBER xxx. (A) INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0198I COMPLEX FORMAT ITEM yyyy IN STATEMENT NUMBER xxx IS INVALID AND HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0202I DEFERRED FEATURE. STATEMENT NUMBER xxx NOT IMPLEMENTED IN THIS VERSION.

Explanation: The statement referred to is of a type not supported by this version of the compiler. For details, refer to Appendix J of this publication.

System Action: Compilation continues

Programmer Response: Probable user error. Rewrite source program avoiding use of unsupported feature. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0207I COMMA REPLACED BY EQUAL SYMBOL IN ASSIGNMENT STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0208I LEFT PARENTHESIS INSERTED IN CHECK LIST IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0209I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO COMPLEX

Explanation: The level of nesting exceeds the implementation restriction. Refer to Appendix J of this publication for details.

System Action: Terminates compilation

Programmer Response: Probable user error. Divide statement into two or more statements. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM0211I LEFT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0212I MULTIPLE TASK OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE FIRST ONE IS USED.

System Action: Ignores options other than the first

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0213I MULTIPLE EVENT OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE FIRST ONE IS USED.

System Action: Ignores options other than the first

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0214I MULTIPLE PRIORITY OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE FIRST ONE IS USED.

System Action: Ignores options other than the first

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0216I INVALID EVENT OPTION IGNORED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0217I INVALID PRIORITY OPTION IGNORED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0218I REPETITION FACTOR MISSING AFTER ITERATION FACTOR IN STATEMENT NUMBER xxx. REPETITION FACTOR OF 1 INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0219I KEYWORD 'CONDITION' NOT SPECIFIED IN SIGNAL STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0220I IDENTIFIER MISSING OR INCORRECT AFTER OPTION IN STATEMENT NUMBER xxx. OPTION DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0221I NUMBER OF LINES NOT GIVEN AFTER LINE OPTION IN STATEMENT NUMBER xxx. (1) INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0222I DEFERRED FEATURE. THE IDENT OPTION ON OPEN/CLOSE STATEMENT NUMBER xxx IS NOT IMPLEMENTED BY THIS VERSION.

Explanation: A language feature has been used that is not supported by this version of the compiler. Refer to Appendix J of this publication for details.

System Action: Option ignored

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0223I EXPRESSION MISSING AFTER IDENT/TITLE/LINESIZE/PAGESIZE OPTION IN STATEMENT NUMBER xxx. OPTION DELETED.

Explanation: No left parenthesis found following keyword

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0224I INVALID OPTION DELETED IN I/O STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0225I OPTION AFTER OPEN/CLOSE IN STATEMENT NUMBER xxx IS INVALID OR MISSING.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0226I EXPRESSION MISSING AFTER FORMAT ITEM IN STATEMENT NUMBER xxx. ITEM DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0227I NO FILE/STRING OPTION SPECIFIED IN ONE OR MORE GET/PUT STATEMENTS. SYSIN/SYSPRINT HAS BEEN ASSUMED IN EACH CASE

Explanation: One or more GET or PUT statements have appeared in the program with no specified FILE option or STRING option.

System Action: The compiler has assumed the appropriate default file (SYSIN for GET, SYSPRINT for PUT).

S IEM0228I EXPRESSION MISSING AFTER OPTION IN STATEMENT NUMBER xxx. OPTION DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0229I FORMAT ITEM IN STATEMENT NUMBER xxx IS INVALID AND HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0230I INVALID DATA LIST IN STATEMENT NUMBER xxx. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0231I MISSING COMMA INSERTED IN DATA LIST IN STATEMENT NUMBER xxx

Explanation: Comma missing between elements of a data list

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0232I KEYWORD DO MISSING IN DATA LIST IN STATEMENT NUMBER xxx. DO IS INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0233I RETURN STATEMENT NUMBER xxx IS WITHIN AN ON-UNIT . IT IS REPLACED BY A NULL STATEMENT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0235I ARGUMENT OMITTED FOLLOWING yyyy OPTION IN STATEMENT NUMBER xxx. OPTION DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0236I THE OPTION yyyy IN STATEMENT NUMBER xxx IS UNSUPPORTED OR INVALID.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0237I INSUFFICIENT OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE STATEMENT HAS BEEN REPLACED BY A NULL STATEMENT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0238I THE LOCATE-VARIABLE IN LOCATE STATEMENT NUMBER xxx IS OMITTED OR SUBSCRIPTED. THE STATEMENT HAS BEEN DELETED.

Explanation: The omission of the locate variable renders the statement meaningless. Subscripted locate variables are invalid.

System Action: Replaces invalid statement with a null statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0240I COMPILER ERROR IN PHASE CV. SCAN CANNOT IDENTIFY DICTIONARY ENTRY.

Explanation: The main scan of fifth pass of read-in has found something in the dictionary which it cannot recognize

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM0241I MULTIPLE USE OF A PREFIX OPTION HAS OCCURRED IN STATEMENT NUMBER xxx. THE LAST NAMED OPTION IS USED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0242I PREFIX OPTION INVALID OR MISSING IN STATEMENT NUMBER xxx. INVALID OPTION DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0243I COMPILER ERROR. PHASE CS HAS FOUND AN UNMATCHED END.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM0244I CHECK PREFIX OPTION IN STATEMENT NUMBER xxx IS NOT FOLLOWED BY A PARENTHESIZED LIST. THE OPTION HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0245I A CHECK PREFIX OPTION IS GIVEN FOR STATEMENT NUMBER xxx WHICH IS NOT A PROCEDURE OR BEGIN. THE OPTION HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0247I ALL SUBSCRIPTED LABELS PREFIXING PROCEDURE OR ENTRY STATEMENT NUMBER xxx HAVE BEEN IGNORED.

Explanation: Subscripted labels may not be used as prefixes on PROCEDURE or ENTRY statements.

Programmer Response: Probable user error. Correct program

and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0254I COMPILER UNABLE TO RECOVER FROM I/O ERROR - PLEASE RETRY JOB.

System Action: Terminates compilation

Programmer Response: Re-attempt compilation. If the input/output error persists, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement was included for the failing job step.
- Have the associated job stream and source program listing available.

T IEM0255I THERE ARE NO COMPLETE STATEMENTS IN THIS PROGRAM

Explanation: Compiler cannot reconcile END statements with stack entries. Usually caused by a program containing only comments.

System Action: Compilation is terminated

Programmer Response: Check source for completed statements. If these are present then do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0257I DATA DIRECTED I/O LIST IN STATEMENT NUMBER xxx CONTAINS BASED ITEM zzzz

System Action: Statement will be deleted by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0258I NUMBER OF SUBSCRIPTS SPECIFIED FOR zzzz IN STATEMENT NUMBER xxx CONFLICTS WITH DIMENSIONALITY. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0510I THE TASK OPTION HAS BEEN ASSUMED TO APPLY TO THE EXTERNAL PROCEDURE STATEMENT NUMBER xxx

Explanation: TASK, EVENT or PRIORITY options have been detected in a CALL statement, but the TASK option has not been specified in the external procedure.

System Action: The TASK option is correctly applied

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0511I OPTIONS MAIN AND/OR TASK ARE NOT ALLOWED ON THE INTERNAL PROCEDURE STATEMENT NUMBER xxx

System Action: The invalid options are ignored

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0512I IDENTIFIER yyyy IN STATEMENT NUMBER xxx IN INITIAL ATTRIBUTE LIST IS NOT A KNOWN LABEL CONSTANT AND HAS BEEN IGNORED.

System Action: Identifier changed to * in the list.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0513I REPEATED LABEL IN SAME BLOCK ON STATEMENT NUMBER xxx. LABEL DELETED.

Explanation: A label may not be used more than once in the same block.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0514I PARAMETER yyyy IN STATEMENT NUMBER xxx IS SAME AS LABEL. PARAMETER REPLACED BY DUMMY.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0515I IMPLEMENTATION RESTRICTION. CHARACTER STRING LENGTH IN STATEMENT NUMBER xxx REDUCED TO 32,767.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0516I ILLEGAL OPTIONS LIST ON STATEMENT NUMBER xxx. LIST IGNORED.

System Action: Compiler scans for next right bracket. If this is not the bracket closing the illegal options list, a compiler error will probably follow.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0517I CONFLICTING ATTRIBUTE DELETED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0518I IMPLEMENTATION RESTRICTION. PRECISION TOO LARGE IN STATEMENT NUMBER xxx . DEFAULT PRECISION GIVEN.

Explanation: If later a valid precision is given, this will be accepted in place of the default.

System Action: Attribute ignored. Attribute test mask restored so that later attribute will not be found to conflict with deleted one.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0519I ILLEGAL ATTRIBUTE ON STATEMENT NUMBER xxx IGNORED.

Explanation: Only data attributes allowed on procedure or entry statements. (No dimensions allowed).

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0520I COMPILER ERROR CODE nnnn

Explanation: A compiler error has occurred.

System Action: Terminates immediately

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0521I INVALID STRING LENGTH IN STATEMENT NUMBER xxx. LENGTH OF 1 ASSUMED.

Explanation: Either no length has been given or string length * has been used in source code.

System Action: Assumes length of 1 and skips to next attribute

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0522I IMPLEMENTATION RESTRICTION. NUMBER OF PARAMETERS IN PROCEDURE OR ENTRY STATEMENT NUMBER xxx TRUNCATED TO 64.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0523I PARAMETER zzzz IN STATEMENT NUMBER xxx APPEARS TWICE. SECOND ONE REPLACED BY DUMMY.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0524I IDENTIFIER yyyy IN LABEL LIST IN STATEMENT NUMBER xxx IS NOT A LABEL OR IS NOT KNOWN.

System Action: Ignores identifier

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0525I IMPLEMENTATION RESTRICTION. TOO MANY PAIRS OF FACTORED ATTRIBUTE BRACKETS FOR THIS SIZE OPTION.

Explanation: Factor bracket table has overflowed.

System Action: Compilation terminated

Programmer Response: Probable user error. Recompile using a SIZE sufficient to provide a larger block size or reduce factoring by expanding declarations. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

W IEM0526I OPTION MAIN HAS NOT BEEN SPECIFIED FOR THE EXTERNAL PROCEDURE STATEMENT NUMBER xxx

S IEM0527I IMPLEMENTATION RESTRICTION. ARRAY BOUND IN STATEMENT NUMBER xxx IS TOO LARGE AND HAS BEEN REPLACED BY THE MAXIMUM PERMITTED VALUE (32767 OR -32768).

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0528I COMPILER ERROR CODE nnnn IN STATEMENT NUMBER xxx

Explanation: Compiler error found in processing a DECLARE statement

System Action: Terminates compilation

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0529I IMPLEMENTATION RESTRICTION. STRUCTURE LEVEL NUMBER IN STATEMENT NUMBER xxx REDUCED TO 255.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0530I IMPLEMENTATION RESTRICTION. TOO MANY LABELS IN LABEL LIST IN STATEMENT NUMBER xxx. THE LABEL zzzz AND ANY FOLLOWING IT HAVE BEEN IGNORED.

Explanation: There is an implementation restriction limiting the number of label constants following the LABEL attribute to 125.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0532I ILLEGAL ASTERISK AS SUBSCRIPT IN DEFINING LIST IN STATEMENT NUMBER xxx . LIST TRUNCATED.

System Action: Compilation continues with truncated iSUB list, possibly causing cascade errors.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0533I IMPLEMENTATION RESTRICTION. I-SUB VALUE IN STATEMENT NUMBER xxx TOO LARGE. REDUCED TO 32.

Explanation: There is an implementation restriction limiting the number of dimensions to a maximum of 32.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0534I IMPLEMENTATION RESTRICTION.

STRING LENGTH IN STATEMENT NUMBER xxx REDUCED TO 32,767.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0536I IDENTIFIER yyyy IN STATEMENT NUMBER xxx IS NOT A LABEL CONSTANT OR IS NOT KNOWN. IT IS IGNORED.

Explanation: Identifiers following the LABEL attribute must be LABEL constants and must be known.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0537I IMPLEMENTATION RESTRICTION. POSITION CONSTANT IN STATEMENT NUMBER xxx REDUCED TO 32,767.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0538I IMPLEMENTATION RESTRICTION. PRECISION SPECIFICATION IN STATEMENT NUMBER xxx TOO LARGE. DEFAULT PRECISION GIVEN.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0539I ILLEGAL NEGATIVE PRECISION IN STATEMENT NUMBER xxx . DEFAULT PRECISION GIVEN.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0540I * BOUNDS ARE MIXED WITH NON-* BOUNDS IN DECLARE STATEMENT NUMBER xxx . ALL THE BOUNDS ARE MADE *.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0541I LOWER BOUND GREATER THAN UPPER BOUND IN DECLARE OR ALLOCATE STATEMENT NUMBER xxx . THE BOUNDS ARE INTERCHANGED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0542I IMPLEMENTATION RESTRICTION. NUMBER OF DIMENSIONS DECLARED TRUNCATED TO 32 IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0543I COMPILER ERROR. ILLEGAL STATEMENT FOUND IN THE DECLARE CHAIN.

Explanation: Compiler error found in scanning chain of DECLARE statements

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM0544I COMPILER ERROR. INITIAL CODE BYTE QF DECLARE STATEMENT IS NEITHER STATEMENT NUMBER NOR STATEMENT LABEL.

Explanation: Compiler error found in first byte of DECLARE statements

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM0545I COMPILER ERROR. ILLEGAL INITIAL CHARACTER TO DECLARED ITEM IN STATEMENT NUMBER xxx

Explanation: Compiler error found in scanning start of declared item

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM0546I COMPILER ERROR. ILLEGAL CHARACTER FOUND AFTER LEVEL NUMBER IN DECLARE STATEMENT NUMBER xxx

Explanation: Compiler error found after structure level number in DECLARE statement

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

W IEM0547I THE IDENTIFIER yyyy DECLARED IN STATEMENT NUMBER xxx IS A NON-MAJOR STRUCTURE MEMBER AND HAS THE SAME NAME AS A FORMAL PARAMETER OR INTERNAL ENTRY POINT. ALL REFERENCES TO THE STRUCTURE MEMBER SHOULD BE QUALIFIED.

System Action: Same BCD treated as different identifiers

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0548I COMPILER ERROR. ILLEGAL CHARACTER FOUND IN DECLARATION LIST.

Explanation: Compiler error found in list of declarations in DECLARE statement

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a

formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

E IEM0549I THE DECLARED LEVEL OF IDENTIFIER yyyy IN STATEMENT NUMBER xxx SHOULD BE ONE. THIS HAS BEEN FORCED.

System Action: Illegal level number treated as 1

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0550I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH A TRUE LEVEL NUMBER GREATER THAN THE IMPLEMENTATION RESTRICTION OF 63. THE DECLARATION OF THE IDENTIFIER IS IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0551I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH ZERO PRECISION. THE DEFAULT VALUE HAS BEEN ASSUMED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0552I COMPILER ERROR. ILLEGAL CHARACTER FOUND IN FACTORED ATTRIBUTE LIST IN DECLARE STATEMENT NUMBER xxx

Explanation: Compiler error found in factored attribute list

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

E IEM0553I THE IDENTIFIER yyyy HAS HAD A CONFLICTING ATTRIBUTE IGNORED IN DECLARE STATEMENT NUMBER xxx

Explanation: The two attributes may conflict as a result of a feature not supported by this version of the compiler. For details of these features, refer to Appendix J of this publication.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0554I COMPILER ERROR. ILLEGAL CHARACTER FOUND IN PARAMETER LIST FOLLOWING 'GENERIC' ATTRIBUTE.

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

E IEM0555I STORAGE CLASS ATTRIBUTES MAY NOT BE SPECIFIED FOR STRUCTURE

MEMBER yyyy . ATTRIBUTE IGNORED.

Programmer Response: Probable user error. Delete illegal storage class attribute for the structure member. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0556I COMPILER ERROR. ILLEGAL CHARACTER FOUND IN PARAMETER LIST FOLLOWING AN 'ENTRY' ATTRIBUTE IN DECLARE STATEMENT NUMBER xxx

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM0557I THE MULTIPLE DECLARATION OF IDENTIFIER yyyy IN STATEMENT NUMBER xxx HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0558I IMPLEMENTATION RESTRICTION. NUMBER OF PARAMETER DESCRIPTIONS DECLARED FOR PROCEDURE OR ENTRY NAME yyyy IN STATEMENT NUMBER xxx TRUNCATED TO 64.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0559I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH CONFLICTING FACTORED LEVEL NUMBERS. THE ONE AT DEEPEST FACTORING LEVEL HAS BEEN CHOSEN.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0560I IN STATEMENT NUMBER xxx A CONFLICTING ATTRIBUTE HAS BEEN IGNORED IN THE DECLARATION OF THE RETURNED VALUE OF ENTRY POINT yyyy

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0561I IN STATEMENT NUMBER xxx THE IDENTIFIER yyyy IS A MULTIPLE DECLARATION OF AN INTERNAL ENTRY LABEL. THIS DECLARATION IS IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0562I THE IDENTIFIER yyyy IS DECLARED IN STATEMENT NUMBER xxx AS AN INTERNAL ENTRY POINT. THE NUMBER OF PARAMETERS DECLARED IS DIFFERENT FROM THE NUMBER GIVEN AT THE ENTRY POINT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0563I THE IDENTIFIER yyyy DECLARED 'BUILTIN' IN STATEMENT NUMBER xxx IS NOT A BUILT-IN FUNCTION. DECLARATION IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0564I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH PRECISION GREATER THAN THE IMPLEMENTATION LIMITS. THE MAXIMUM VALUE HAS BEEN TAKEN.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0565I THE IDENTIFIER yyyy IS DECLARED IN STATEMENT NUMBER xxx AS A MEMBER OF A GENERIC LIST, BUT ITS ATTRIBUTES DO NOT MAKE IT AN ENTRY POINT. THE DECLARATION OF THE IDENTIFIER HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0566I ONE OF THE PARAMETERS DECLARED FOR ENTRY POINT yyyy IN STATEMENT NUMBER xxx SHOULD BE AT LEVEL ONE. THIS HAS BEEN FORCED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0567I IF FUNCTION yyyy IN STATEMENT NUMBER xxx IS INVOKED, THE DEFAULT ATTRIBUTES ASSUMED FOR THE VALUE RETURNED WILL CONFLICT WITH THE ATTRIBUTES IN THE PROCEDURE OR ENTRY STATEMENT FOR THAT VALUE.

Explanation: The data type to which a result will be converted at a RETURN (expression) will not be the same as that expected at an invocation of the entry label as a function.

System Action: None

Programmer Response: Probable user error. Write an entry-point declaration (using the ENTRY or RETURNS attribute) in the containing block, giving the same data attributes as those on the PROCEDURE or ENTRY statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0568I THE IDENTIFIER zzzz IS CALLED BUT IS EITHER A BUILTIN FUNCTION OR IS NOT AN ENTRY POINT.

System Action: The erroneous statement is deleted.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0569I COMPILER ERROR NUMBER nnnn IN MODULE EP.

Explanation: Compiler error found in scan of chain of CALL statements

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the

comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

W IEM0570I THE ENTRY POINT yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx TO HAVE A RETURNED VALUE DIFFERENT FROM THAT GIVEN ON THE PROCEDURE OR ENTRY STATEMENT.

System Action: None

Programmer Response: Probable user error. Change the declaration, or the PROCEDURE or ENTRY statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0571I IMPLEMENTATION RESTRICTION. IDENTIFIER yyyy IN STATEMENT NUMBER xxx HAS MORE THAN 32 DIMENSIONS. DIMENSION ATTRIBUTE IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0572I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH THE ATTRIBUTE "NORMAL" OR "ABNORMAL". THE APPLICATION OF THIS ATTRIBUTE IS AN UNSUPPORTED FEATURE OF THE FOURTH VERSION, AND IT HAS BEEN IGNORED.

Explanation: A language feature has been used which is not supported by this version of the compiler. Refer to Appendix J of this publication for details.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0573I THE SELECTION OF GENERIC FAMILY MEMBERS WHOSE PARAMETERS HAVE A STRUCTURE DESCRIPTION IS DEFERRED. ENTRY NAME yyyy, DECLARED IN STATEMENT NUMBER xxx, IS SUCH A MEMBER AND HAS BEEN DELETED.

Explanation: The usage referred to is not supported by this version of the compiler. For details, refer to Appendix J.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0574I THE MULTIPLE DECLARATION OF IDENTIFIER yyyy IN STATEMENT NUMBER xxx HAS BEEN IGNORED.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

W IEM0575I THE REPEATED ATTRIBUTE IN THE DECLARATION OF FILE yyyy IN STATEMENT NUMBER xxx HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0576I THE EXTERNAL FILE yyy DECLARED IN STATEMENT NUMBER xxx HAS THE SAME NAME AS THE EXTERNAL PROCEDURE, DECLARATION IGNORED.

Programmer Response: Probable user error. Correct program

and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0577I INCORRECT SPECIFICATION OF THE ARRAY BOUNDS, STRING LENGTH OR AREA SIZE OF THE NON-CONTROLLED PARAMETER yyyy IN DECLARE STATEMENT NUMBER xxx; THOSE OF THE CORRESPONDING ARGUMENT WILL BE ASSUMED.

Explanation: In the declaration of a non-controlled array, string or area parameter, the bounds, length or size must be given by a constant or by an asterisk.

System Action: The bounds, length or size of the array, string or area passed as an argument will be taken.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0578I UNDIMENSIONED VARIABLE yyy DECLARED IN STATEMENT NUMBER xxx HAS INITIAL ATTRIBUTE WITH CONFLICTING SPECIFICATION FOR A DIMENSIONED VARIABLE. INITIAL ATTRIBUTE IGNORED.

Programmer Response: Probable user error. Correct source program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0579I THE PARAMETER OF THE MAIN PROCEDURE SHOULD BE A FIXED LENGTH CHARACTER STRING OR HAVE THE ATTRIBUTES CHARACTER(100) VARYING.

Programmer Response: Probable user error. Correct source program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0580I INVALID USE OF FILE yyy IN STATEMENT NUMBER xxx. IT HAS BEEN REPLACED BY A DUMMY REFERENCE.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0589I COMPILER ERROR. ITEM zzzz IN LIKE CHAIN IS NOT A STRUCTURE. ITEM IS IGNORED.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,EV)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0590I STRUCTURE ELEMENT zzzz WHICH HAS LIKE ATTRIBUTE ATTACHED TO IT, IS FOLLOWED BY AN ELEMENT WITH A NUMERICALLY GREATER STRUCTURE LEVEL NUMBER. LIKE ATTRIBUTE IS IGNORED.

System Action: Self explanatory: may result in cascade errors.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0591I STRUCTURE ELEMENT zzzz IS LIKENED TO AN ITEM WHICH IS NOT A STRUCTURE VARIABLE. LIKE ATTRIBUTE IS IGNORED.

System Action: Self explanatory: may result in cascade errors.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0592I STRUCTURE ELEMENT zzzz IS LIKENED TO A STRUCTURE WHICH CONTAINS ELEMENTS WHICH HAVE ALSO BEEN DECLARED WITH THE LIKE ATTRIBUTE. LIKE ATTRIBUTE ON ORIGINAL STRUCTURE IS IGNORED.

System Action: Self explanatory: may result in cascade errors.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0593I STRUCTURE NAME TO WHICH zzzz IS LIKENED IS NOT KNOWN. LIKE ATTRIBUTE IGNORED.

System Action: Self explanatory: may result in cascade errors.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0594I AMBIGUOUS QUALIFIED NAME yyyy USED AS A BASE IDENTIFIER. MOST RECENT DECLARATION USED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0595I QUALIFIED NAME yyyy USED AS A BASE IDENTIFIER CONTAINS MORE THAN ONE IDENTIFIER AT THE SAME STRUCTURE LEVEL.

System Action: The erroneous statement is deleted.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0596I MAJOR STRUCTURE yyyy HAS BEEN LIKENED TO AN ITEM WHICH IS NOT A VALID STRUCTURE. DECLARATION OF STRUCTURE IGNORED.

System Action: Self explanatory: may result in cascade errors.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0597I IDENTIFIER zzzz WHICH IS NOT A FORMAL PARAMETER OR OF STORAGE CLASS CONTROLLED HAS BEEN LIKENED TO A STRUCTURE CONTAINING * DIMENSIONS OR LENGTH. * DIMENSIONS OR LENGTH HAVE BEEN IGNORED IN THE CONSTRUCTED STRUCTURE.

System Action: Self explanatory: may result in cascade errors from later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0598I QUALIFIED NAME TO WHICH zzzz HAS BEEN LIKENED IS AN AMBIGUOUS REFERENCE. LIKE ATTRIBUTE HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0599I zzzz WHICH IS A PARAMETER OR A BASED VARIABLE, HAS BEEN DECLARED (USING THE LIKE ATTRIBUTE) AS A STRUCTURE WITH THE INITIAL ATTRIBUTE. THE INITIAL ATTRIBUTE IS INVALID AND HAS BEEN IGNORED.

Programmer Response: Probable user error. Declare the parameter or based variable with the LIKE attribute specifying a structure without the INITIAL attribute. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0600I STATIC STRUCTURE zzzz HAS BEEN DECLARED BY MEANS OF THE LIKE ATTRIBUTE TO HAVE ADJUSTABLE EXTENTS. THE EXTENTS HAVE BEEN IGNORED.

Explanation: A STATIC variable cannot have adjustable extents

System Action: All bounds on the offending variable are set to zero

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0601I OFFSET ATTRIBUTE ON PROCEDURE STATEMENT NUMBER xxx IS NOT BASED ON A BASED AREA. IT HAS BEEN CHANGED TO POINTER.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0602I IDENTIFIER IN BASED ATTRIBUTE ON zzzz DECLARED IN STATEMENT NUMBER xxx IS NOT A NON-BASED POINTER

System Action: Compilation is terminated

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM0603I INVALID POINTER EXPRESSION IN BASED ATTRIBUTE ON zzzz IN STATEMENT NUMBER xxx

Explanation: The pointer associated with the based variable does not obey the implementation rules (e.g., it may be subscripted).

System Action: The compilation is terminated

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM0604I LENGTH OR SIZE DECLARED FOR BASED STRING OR BASED AREA zzzz IN STATEMENT NUMBER xxx IS INVALID

Explanation: The declaration violates the compiler implementation rules. (See Appendix J, 'Implementation Conventions and Restrictions').

System Action: Terminates compilation

Programmer Response: Probable user error. If the problem

recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM0605I BOUNDS DECLARED FOR BASED ARRAY zzzz IN STATEMENT NUMBER xxx ARE INVALID

Explanation: The adjustable bounds declared are outside those permitted by this implementation.

System Action: Terminates compilation

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0606I OFFSET VARIABLE zzzz HAS BEEN DECLARED IN STATEMENT NUMBER xxx RELATIVE TO AN IDENTIFIER WHICH IS NOT A LEVEL 1 BASED AREA. IT HAS BEEN CHANGED TO A POINTER VARIABLE.

System Action: The offset is changed to a pointer to prevent the compiler from producing further error messages.

Programmer Response: Probable user error. Consider assigning the ADDR of the required area to the pointer named in the declaration of a level 1 based area; this area can be validly named in the OFFSET attribute, and offset values for it will be correct for the other area. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0607I IF THE BASE OF zzzz CORRESPONDENCE DEFINED IN STATEMENT NUMBER xxx IS ALLOCATED WITH THE DECLARED BOUNDS THE DEFINING WILL BE IN ERROR.

Explanation: For correspondence defining not involving iSUB's, the bounds of the defined array must be a subset of the bounds of the base. In this case the bounds declared for the base do not satisfy this requirement. However, the base is of CONTROLLED storage class and if it is allocated with different bounds the defining may be legal.

System Action: Nothing further

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0608I ILLEGAL DEFINING IN STATEMENT NUMBER xxx. BASE IDENTIFIER zzzz IS A MEMBER OF A DIMENSIONED STRUCTURE.

Explanation: In the case of string class overlay defining where the base is an array, it is an error if it is a member of an array of structures.

System Action: The compilation is terminated.

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - "The DEFINED Attribute" - and correct error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the

comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM0609I DEFERRED FEATURE. DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx WITH A SUBSCRIPTED BASE.

Explanation: Overlay defining on a subscripted base is not supported by this version of the compiler.

System Action: The compilation is terminated.

Programmer Response: Probable user error. Replace all references to the defined item by appropriate subscripted references to the base. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM0610I DEFERRED FEATURE. DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx ON A BASE OF CONTROLLED STORAGE CLASS.

Explanation: If the base is declared CONTROLLED, neither overlay defining nor correspondence defining is supported by this release of the compiler.

System Action: The compilation is terminated.

Programmer Response: Probable user error. Replace all references to the defined item by appropriate references to the base. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM0611I SCALAR zzzz DECLARED IN STATEMENT NUMBER xxx IS ILLEGALLY DEFINED WITH ISUBS.

Explanation: Only arrays may be correspondence defined using iSUB notation.

System Action: The compilation is terminated.

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - "The DEFINED Attribute" - and correct error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM0612I INITIAL ATTRIBUTE DECLARED FOR DEFINED ITEM zzzz IN STATEMENT NUMBER xxx WILL BE IGNORED.

Explanation: DEFINED items may not have the INITIAL attribute.

System Action: INITIAL attribute ignored

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0613I ELEMENT VARIABLE SPECIFIED IN REFER OPTION IN STATEMENT NUMBER xxx IS NOT AN INTEGER.

Explanation: Both I and N in (I REFER(N)) must be fixed

binary integers; they must also be of the same precision.

System Action: Compilation continues. Any reference to the element variable may result in an execution error.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0614I ELEMENT VARIABLES SPECIFIED IN REFER OPTION IN STATEMENT NUMBER xxx DO NOT HAVE THE SAME PRECISION.

Explanation: Both I and N in (I REFER(N)) must be fixed binary integers of the same precision.

System Action: Compilation continues. Any reference to either element variable may result in an execution error.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0623I THE BASE SUBSCRIPT LIST USED WITH THE DEFINED VARIABLE zzzz IN STATEMENT NUMBER xxx ILLEGALLY REFERS TO OR IS DEPENDENT ON THE DEFINED VARIABLE.

Explanation: It is illegal for a base subscript list in the DEFINED attribute to refer directly, or via any further level of defining, to the defined item.

System Action: The compilation is terminated.

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - "The DEFINED Attribute" - and correct error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM0624I THE BASE IDENTIFIER FOR zzzz DECLARED IN STATEMENT NUMBER xxx IS DEFINED OR BASED.

Explanation: The base of DEFINED data may not itself be DEFINED.

System Action: Compilation terminated

Programmer Response: Probable user error. Replace the specified base by an appropriate reference to its base. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM0625I THE DEFINING BASE FOR zzzz DECLARED IN STATEMENT NUMBER xxx HAS THE WRONG NUMBER OF SUBSCRIPTS.

Explanation: If the base reference in a DEFINED attribute is subscripted, it must have the same number of subscript expressions as the dimensionality of the base array.

System Action: The compilation is terminated.

Programmer Response: Probable user error. Correct the subscript list, or declaration of the base, whichever is appropriate. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM0626I THE DEFINING BASE FOR zzzz DECLARED IN STATEMENT NUMBER xxx IS NOT DATA.

Explanation: The only legal data types that may be used for defining bases are String, Arithmetic, Task, Event, and Label.

System Action: The compilation is terminated.

Programmer Response: Probable user error. Check that the defining base is correctly written and declared. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM0628I IMPLEMENTATION RESTRICTION. THE NESTING OF REFERENCES TO DATA DEFINED WITH A SUBSCRIPTED BASE IS TOO DEEP.

Explanation: The complexity of defining has resulted in a level of nesting which is too great for the compiler.

System Action: The compilation is terminated.

Programmer Response: Probable user error. Reduce complexity of defining. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM0629I ARRAY zzzz DECLARED IN STATEMENT NUMBER xxx ILLEGALLY HAS THE POS ATTRIBUTE WITH ISUB DEFINING.

Explanation: The POS attribute may not be specified for correspondence defining.

System Action: The compilation is terminated.

Programmer Response: Probable user error. Delete POS attribute If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM0630I THE DESCRIPTION OF zzzz CORRESPONDENCE DEFINED IN STATEMENT NUMBER xxx DOES NOT MATCH THAT OF THE DEFINING BASE.

Explanation: For correspondence defining, if either the base or the defined item are arrays of structures, then both must be arrays of structures.

System Action: The compilation is terminated.

Programmer Response: Probable user error. Correct the program. Note that POS (1) may be used to force overlay defining. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a

formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM0631I IMPLEMENTATION RESTRICTION. THE CORRESPONDENCE DEFINING OF yyyy AN ARRAY OF STRUCTURES DECLARED IN STATEMENT NUMBER xxx.

Explanation: Correspondence defining with arrays of structures is not supported by the compiler.

System Action: The compilation is terminated.

Programmer Response: Probable user error. Declare the base arrays of the defined structure as correspondence defined on the matching base arrays of the base structure. Note that for this to be valid, the base arrays of the defined structure must have unique names and be declared at level 1. This alternative method precludes structure operations with the defined item, but achieves the desired mapping. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM0632I THE BOUNDS OF zzzz CORRESPONDENCE DEFINED IN STATEMENT NUMBER xxx ARE NOT A SUBSET OF THE BASE.

Explanation: For correspondence defining not involving iSUB's, the bounds of the defined array must be a subset of the corresponding bounds of the base array.

System Action: The compilation is terminated.

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - "The DEFINED Attribute" - and correct program. Note that POS (1) may be used to force overlay defining. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

S IEM0633I ITEM TO BE ALLOCATED IN STATEMENT NUMBER xxx IS NOT AT LEVEL 1. THE STATEMENT HAS BEEN IGNORED.

Explanation: An identifier specified in an ALLOCATE statement must refer to a major structure or data not contained in a structure. A major structure identifier may optionally be followed by a full structure description.

System Action: The ALLOCATE statement is deleted

Programmer Response: Probable user error. Replace erroneous identifier by that of the containing major structure. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0634I ITEM TO BE ALLOCATED IN STATEMENT NUMBER xxx HAS NOT BEEN DECLARED. THE STATEMENT HAS BEEN IGNORED.

Explanation: Only CONTROLLED data may be allocated. Data may only obtain the attribute CONTROLLED from an explicit declaration.

System Action: The ALLOCATE statement is deleted

Programmer Response: Probable user error. Construct a DECLARE statement for the

identifier. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0636I ITEM TO BE ALLOCATED IN STATEMENT NUMBER xxx WAS NOT DECLARED CONTROLLED. THE STATEMENT HAS BEEN IGNORED.

Explanation: Only CONTROLLED data may be specified in ALLOCATE statements.

System Action: The ALLOCATE statement is deleted

Programmer Response: Probable user error. Declare the identifier CONTROLLED. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0637I A CONFLICTING ATTRIBUTE WAS GIVEN FOR zzzz IN STATEMENT NUMBER xxx. THE ATTRIBUTE HAS BEEN IGNORED.

Explanation: Attributes given for an identifier in an ALLOCATE statement may not conflict with those given explicitly or assumed by default from the declaration.

System Action: Ignores the attribute from the ALLOCATE

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0638I THE STRUCTURE DESCRIPTION GIVEN ON STATEMENT NUMBER xxx DIFFERS FROM THAT DECLARED. THE STATEMENT HAS BEEN IGNORED.

Explanation: If a description of a major structure is given on an ALLOCATE statement, the description must match that declared.

System Action: The ALLOCATE statement is deleted

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0640I AN INVALID ATTRIBUTE WAS GIVEN IN STATEMENT NUMBER xxx. THE STATEMENT HAS BEEN IGNORED.

Explanation: Only CHAR, BIT, INITIAL, and Dimension attributes are permitted in ALLOCATE statements.

System Action: The ALLOCATE statement is deleted

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0641I CONFLICTING ATTRIBUTES HAVE BEEN GIVEN FOR zzzz IN STATEMENT NUMBER xxx. THE FIRST LEGAL ONE HAS BEEN USED.

Explanation: At most, one attribute in the following classes may be given for an identifier in an ALLOCATE statement: Dimension, String(CHAR or BIT), INITIAL.

System Action: All attributes after the first in a particular class are ignored.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0642I DIMENSIONALITY GIVEN IN STATEMENT NUMBER xxx DIFFERS FROM THAT DECLARED. THE STATEMENT HAS BEEN IGNORED.

Explanation: If a dimension attribute is given for an identifier in an ALLOCATE statement, the identifier must

have been declared with the same dimensionality.

System Action: The ALLOCATE statement is deleted

Programmer Response: Probable user error. Correct declaration or ALLOCATE Statement, whichever applicable. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0643I THE LEVEL NUMBER DECLARED FOR zzzz IS NOT THE SAME AS THAT GIVEN IN STATEMENT NUMBER xxx. THE FORMER HAS BEEN USED.

Explanation: If a structure description is given in an ALLOCATE Statement, it must match the declaration. The indicated level number discrepancy may be an error.

System Action: Nothing further

Programmer Response: Probable user error. Check that ALLOCATE statement is as intended. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0644I STATEMENT NUMBER xxx CONTAINS AN ILLEGAL PARENTHESIZED LIST. THE STATEMENT HAS BEEN IGNORED.

Explanation: Factored attribures are not allowed on ALLOCATE statements.

System Action: Statement ignored

Programmer Response: Probable user error. Remove parentheses and any factored attributes. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0645I ATTRIBUTE GIVEN WITH BASED VARIABLE zzzz IN ALLOCATE STATEMENT NUMBER xxx HAS BEEN IGNORED.

Explanations: Based variable may not be specified with attributes

Programmer Response: Probable user error. Correct ALLOCATE statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0646I IDENTIFIER yyyy PRECEDING POINTER QUALIFIER IN STATEMENT NUMBER xxx IS NOT A NON-BASED POINTER VARIABLE

System Action: The identifier is replaced by a dummy dictionary reference; a later phase will delete the statement.

Programmer Response: Probable user error. Correct the invalid statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0647I POINTER-QUALIFIED IDENTIFIER zzzz IN STATEMENT NUMBER xxx IS NOT A BASED VARIABLE

System Action: Identifier is replaced by a dummy dictionary reference; a later phase will delete the statement.

Programmer Response: Probable user error. Correct the invalid statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0653I COMPILER ERROR. ILLEGAL ENTRY IN STATEMENT NUMBER xxx

Explanation: Compiler error found in scan of statement

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

S IEM0655I QUALIFIED NAME BEGINNING yyyy USED IN STATEMENT NUMBER xxx BUT NO PREVIOUS STRUCTURE DECLARATION GIVEN. DUMMY REFERENCE INSERTED.

System Action: Reference to the illegal variable or the whole statement will be deleted by later phases.

Programmer Response: Probable user error. Correct program by inserting DECLARE statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0656I MOST RECENT DECLARATION USED OF AMBIGUOUS QUALIFIED NAME OR STRUCTURE MEMBER BEGINNING yyyy IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0657I QUALIFIED NAME BEGINNING yyyy IN STATEMENT NUMBER xxx CONTAINS MORE THAN ONE IDENTIFIER AT THE SAME STRUCTURE LEVEL.

System Action: The statement is deleted

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0658I QUALIFIED NAME BEGINNING yyyy IN STATEMENT NUMBER xxx IS AN AMBIGUOUS REFERENCE. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by later phase

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0659I UNSUPPORTED FEATURE. STRING PSEUDO-VARIABLE APPEARS IN REPLY, KEYTO OR STRING OPTION IN STATEMENT NUMBER xxx. STATEMENT WILL BE DELETED BY A LATER PHASE.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0660I PSEUDO-VARIABLE yyyy IN STATEMENT NUMBER xxx IS INVALID BECAUSE IT IS NESTED IN ANOTHER PSEUDO-VARIABLE.

Explanation: Language restriction. Pseudo-variables cannot be nested.

System Action: The statement will be deleted by a later phase.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0661I INVALID USE OF THE PSEUDO-VARIABLE 'PRIORITY' IN STATEMENT NUMBER xxx. THE TASK OPTION HAS NOT BEEN SPECIFIED IN THE EXTERNAL PROCEDURE.

Explanation: The PRIORITY pseudo-variable can be used only in a multitasking environment.

System Action: The statement is deleted.

Programmer Response: Probable user error. Recompile with the TASK option in the external procedure options list. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0662I IN STATEMENT NUMBER xxx CHECK LIST CONTAINS DEFINED ITEM yyy. IT HAS BEEN REPLACED BY ITS BASE IDENTIFIER.

Explanation: Defined items are not permitted in CHECK lists.

System Action: The base item is assumed to replace the reference to the defined item in the CHECK list. References in the text to the defined item will not be checked.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0673I INVALID USE OF FUNCTION NAME ON LEFT HAND SIDE OF EQUAL SYMBOL, OR IN REPLY KEYTO OR STRING OPTION, IN STATEMENT NUMBER xxx

System Action: Statement will be deleted by later phases

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0674I STATEMENT NUMBER xxx CONTAINS ILLEGAL USE OF FUNCTION yyyy

System Action: Reference to function or whole statement will be deleted by later phases

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0675I IN STATEMENT NUMBER xxx IDENTIFIER yyyy AFTER GO TO IS NOT A LABEL OR LABEL VARIABLE KNOWN IN THE BLOCK CONTAINING THE GO TO.

System Action: Statement deleted by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0676I DEFERRED FEATURE. IDENTIFIER yyyy NOT ALLOWED AS A BUILT-IN FUNCTION OR PSEUDO-VARIABLE . DUMMY REFERENCE INSERTED IN STATEMENT NUMBER xxx

Explanation: A language feature has been used that is not supported by this version of the compiler. For details, refer to Appendix J of this publication.

System Action: Statement deleted by later phases

Programmer Response: Probable user error. Correct statement by removing reference to function in error. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0677I ILLEGAL PARENTHESIZED LIST IN STATEMENT NUMBER xxx FOLLOWS AN IDENTIFIER WHICH IS NOT A FUNCTION OR ARRAY. LIST DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0682I IN STATEMENT NUMBER xxx GO TO TRANSFERS CONTROL ILLEGALLY TO A FORMAT STATEMENT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0683I zzzz WAS FOUND WHERE A FILENAME IS REQUIRED IN STATEMENT NUMBER xxx. DUMMY DICTIONARY REFERENCE REPLACES ILLEGAL ITEM.

System Action: Statement will be deleted by later phases

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0684I USE OF LABEL VARIABLE zzzz MAY RESULT IN AN ILLEGAL BRANCH IN STATEMENT NUMBER xxx

Explanation: It is possible that the label variable may contain a value which would cause control to branch illegally into a block.

System Action: None

Programmer Response: Probable user error. Check validity of possible branches. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0685I zzzz IS NOT A STATEMENT LABEL ON AN EXECUTABLE STATEMENT. DUMMY REFERENCE INSERTED AFTER GO TO IN STATEMENT NUMBER xxx

System Action: Statement will be deleted by later phases

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0686I zzzz APPEARS IN A FREE OR ALLOCATE STATEMENT BUT HAS NOT BEEN DECLARED CONTROLLED. DUMMY REFERENCE INSERTED IN STATEMENT NUMBER xxx.

System Action: A dummy reference is inserted. The statement will be deleted by a later phase

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0687I IN STATEMENT NUMBER xxx DICTIONARY REFERENCE zzzz TRANSFERS CONTROL ILLEGALLY TO ANOTHER BLOCK OR GROUP. EXECUTUION ERRORS MAY OCCUR.

S IEM0688I COMPILER ERROR. TOO FEW LEFT PARENTHESES IN STATEMENT NUMBER xxx

Explanation: This is a compiler error

System Action: None taken. Cascade errors may result.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,FI)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0689I zzzz WAS FOUND WHERE A TASK IDENTIFIER IS REQUIRED IN STATEMENT NUMBER xxx. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by later phases

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0690I zzzz WAS FOUND WHERE EVENT VARIABLE IS REQUIRED IN STATEMENT NUMBER xxx. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by later phases

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0691I INVALID ITEM zzzz IN DATA LIST, OR 'FROM' OR 'INTO' OPTION, IN STATEMENT NUMBER xxx

System Action: Statement will be deleted by later phases

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0692I DATA DIRECTED I/O LIST OR FROM OR INTO OPTION IN STATEMENT NUMBER xxx CONTAINS A PARAMETER, DEFINED OR BASED ITEM zzzz.

System Action: Statement will be deleted by later phases

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0693I ILLEGAL USE OF FUNCTION zzzz IN INPUT LIST IN STATEMENT NUMBER xxx. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by later phases

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0694I IN THE FORMAT LIST IN STATEMENT NUMBER xxx A REMOTE FORMAT ITEM REFERENCES zzzz, WHICH IS NOT A STATEMENT LABEL IN THE CURRENT BLOCK. DUMMY REFERENCE INSERTED.

System Action: Format item deleted by later phase

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0695I LABEL ARRAY zzzz IS NOT FOLLOWED BY A SUBSCRIPT LIST AFTER GO TO IN STATEMENT NUMBER xxx. DUMMY REFERENCE REPLACES REFERENCE TO ARRAY.

System Action: Statement will be deleted by later phases

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0696I IN STATEMENT NUMBER xxx IT IS AN ERROR IF THE PARAMETER zzzz IN A REMOTE FORMAT ITEM REFERS TO A FORMAT STATEMENT WHICH IS NOT INTERNAL TO THE SAME BLOCK AS THE REMOTE FORMAT ITEM.

Explanation: Remote formats become executable code, but not internal procedures. Therefore they must appear in the same block in which they are used.

System Action: Object-time error message is compiled

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0697I STATEMENT LABEL zzzz ATTACHED TO STATEMENT NUMBER xxx IS USED AS A REMOTE FORMAT ITEM IN THAT STATEMENT. A DUMMY REPLACES THE REMOTE FORMAT ITEM.

System Action: Statement will be deleted by a later phase

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0698I THE BASED VARIABLE zzzz IN LOCATE STATEMENT NUMBER xxx IS NOT AT LEVEL 1. DUMMY REFERENCE INSERTED.

System Action: The statement will be deleted by a later phase.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0699I STRUCTURE ARGUMENT zzzz OF FROM OR INTO OPTION IN STATEMENT NUMBER xxx IS NOT A MAJOR STRUCTURE. DUMMY REFERENCE INSERTED

System Action: Statement deleted by later phase

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0700I ILLEGAL USE OF FUNCTION, LABEL OR VARYING STRING zzzz AS ARGUMENT OF FROM OR INTO OPTION IN STATEMENT NUMBER xxx. DUMMY REFERENCE INSERTED.

System Action: Statement deleted by later phase

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0701I ARGUMENT zzzz OF SET OPTION IS NOT A POINTER VARIABLE. DUMMY REFERENCE INSERTED.

System Action: Statement deleted by later phase.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0702I LABEL, TASK OR EVENT VARIABLE zzzz USED IN FROM OR INTO OPTION IN STATEMENT NUMBER xxx MAY LOSE ITS VALIDITY IN TRANSMISSION

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0703I INVALID IDENTIFIER zzzz FREED IN STATEMENT NUMBER xxx

Explanation: The identifier in the FREE statement is not:

1. A BASED or a CONTROLLED variable, or
2. A major structure with the BASED or CONTROLLED attribute.

System Action: Invalid identifier replaced by dummy.

Programmer Response: Probable

user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0704I STATEMENT NUMBER xxx CONTAINS INVALID USE OF FUNCTION zzzz

System Action: Statement deleted by later phase

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0705I IF THE LABEL VARIABLE IN GO TO STATEMENT NUMBER xxx ASSUMES THE VALUE OF ITS VALUE-LIST MEMBER zzzz, THE STATEMENT WILL CONSTITUTE AN INVALID BRANCH INTO AN ITERATIVE DO GROUP.

System Action: None

Programmer Response: Probable user error. Check that branch will be valid at execution time. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0706I VARIABLE zzzz IN LOCATE STATEMENT IS NOT A BASED VARIABLE. DUMMY REFERENCE INSERTED.

System Action: The statement is deleted by a later phase.

Programmer Response: Probable user error. Correct the invalid statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0707I ARGUMENT zzzz OF IN OPTION IS NOT AN AREA VARIABLE. DUMMY REFERENCE INSERTED.

System Action: The statement is deleted by a later phase.

Programmer Response: Probable user error. Correct the invalid statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0715I TEXT yyyy ASSOCIATED WITH THE INITIAL ATTRIBUTE IN STATEMENT NUMBER xxx IS ILLEGAL AND HAS BEEN IGNORED.

Explanation: The INITIAL attribute has been used incorrectly.

System Action: The INITIAL attribute is deleted

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0718I INVALID CHECK LIST IN STATEMENT NUMBER xxx. STATEMENT HAS BEEN CHANGED TO 'ON ERROR'.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0719I ELEMENT OF LABEL ARRAY zzzz WHICH IS DECLARED WITH INITIAL ATTRIBUTE USED AS STATEMENT LABEL ON STATEMENT NUMBER xxx

System Action: Label is deleted

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0720I SUBSCRIPTED IDENTIFIER zzzz USED AS LABEL ON STATEMENT NUMBER xxx IS NOT A LABEL ARRAY

System Action: Label is deleted

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0721I ELEMENT OF STATIC LABEL ARRAY zzzz USED AS LABEL ON STATEMENT NUMBER xxx

System Action: Label is deleted

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0722I ELEMENT OF LABEL ARRAY zzzz USED AS LABEL ON STATEMENT NUMBER xxx IN BLOCK OTHER THAN THE ONE IN WHICH IT IS DECLARED.

System Action: An error statement is inserted in the text in place of the offending label.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0723I COMPILER ERROR IN STATEMENT NUMBER xxx

Explanation: Compiler error found in scan of text

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0724I FORMAL PARAMETER zzzz IN CHECK LIST. PARAMETER IS IGNORED.

Explanation: The identifier list of a CHECK prefix must not contain formal parameters

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0725I STATEMENT NUMBER xxx HAS BEEN DELETED DUE TO A SEVERE ERROR NOTED ELSEWHERE.

System Action: The whole statement is replaced by an error statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0726I IDENTIFIER zzzz IN STATEMENT NUMBER xxx IS NOT A FILE NAME. THE STATEMENT IS DELETED.

Explanation: The identifier has been used previously in a different context and is therefore not recognized as a file name.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0727I IDENTIFIER zzzz IN STATEMENT NUMBER xxx IS NOT A CONDITION NAME. THE STATEMENT IS DELETED.

Explanation: The identifier has been used previously in a different context and is therefore not recognized as a condition name

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0728I COMPILATION TERMINATED DUE TO A PREVIOUSLY DETECTED SEVERE ERROR IN STATEMENT NUMBER xxx

Explanation: A previous module has inserted a dummy dictionary reference into the second file. The compiler cannot recover.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0729I COMPILER ERROR IN SCALE FACTOR IN PICTURE BEGINNING yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture halted. All references to picture deleted.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,FO)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0730I MORE THAN ONE SIGN CHARACTER PRESENT IN A SUBFIELD OF PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0731I PICTURE CHARACTER M APPEARS IN NON-STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0732I FIELD MISSING IN STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0733I ILLEGAL EDIT CHARACTERS AT START OF STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0734I ILLEGAL CHARACTER OR ILLEGAL NUMBER OF CHARACTERS IN POUNDS FIELD OF STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0735I ILLEGAL CHARACTER OR ILLEGAL NUMBER OF CHARACTERS IN SHILLINGS FIELD OF STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0736I WRONG NUMBER OF DELIMITER CHARACTERS M IN STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0737I ILLEGAL CHARACTER OR ILLEGAL NUMBER OF CHARACTERS IN PENCE FIELD OF STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0739I STATIC PICTURE CHARACTER $ S + - NOT AT EXTREMITY OF SUBFIELD. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0740I MULTIPLE USE OF E K OR V IN PICTURE. PICTURE TRUNCATED AT ILLEGAL CHARACTER. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Picture truncated at point indicated

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before

calling IBM for programming support:

- Have the source program listing available.

E IEM0741I CR OR DB INCORRECTLY POSITIONED IN SUBFIELD. PICTURE TRUNCATED AT THIS POINT. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Picture truncated at point indicated

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0742I CR OR DB GIVEN FOR A NON-REAL, NON-NUMERIC OR FLOATING FIELD. PICTURE TRUNCATED BEFORE CR OR DB IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Picture truncated at point indicated

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0745I ILLEGAL USE OF PICTURE CHARACTER Z OR * IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0746I STERLING MARKER FOUND IN OTHER THAN FIRST POSITION IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0747I STERLING PICTURE CHARACTERS FOUND IN NON-STERLING PICTURE. SCANNING OF PICTURE STOPPED. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0748I ILLEGAL USE OF SCALING FACTOR IN PICTURE. SCALING FACTOR ONWARDS DELETED. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Picture truncated at point indicated

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0749I ILLEGAL USE OF SCALING FACTOR IN PICTURE. SCAN OF PICTURE TERMINATED. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS

IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0750I ILLEGAL CHARACTER PRESENT IN CHARACTER STRING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0751I NO MEANINGFUL CHARACTERS IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0752I ILLEGAL USE OF,OR ILLEGAL CHARACTERS IN, STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0754I ILLEGAL CHARACTER IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0755I ILLEGAL USE OF DRIFTING EDITING SYMBOLS S $ + - IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0756I IMPLEMENTATION RESTRICTION. PRECISION TOO LARGE OR PICTURE TOO LONG IN PICTURE BEGINNING yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

Programmer Response: Probable user error. Correct program

and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0758I COMPILER ERROR IN PHASE FT.

Explanation: Compiler error found in scan of dictionary

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0759I IMPLEMENTATION RESTRICTION. STERLING CONSTANT EXCEEDS 4166666666666.13.3L. HIGH ORDER DIGITS LOST DURING CONVERSION TO DECIMAL.

System Action: High order digits lost somewhere in the following conversion process: shift pounds field left one digit, double by addition. Add shillings field. Add result, doubled by addition, to result shifted left one digit. Add pence field.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0760I IMPLEMENTATION RESTRICTION. EXPONENT FIELD TOO LARGE IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Reference to picture deleted

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0761I PICTURE CHARACTER E OR K APPEARS WITHOUT AN EXPONENT FIELD FOLLOWING IT. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Reference to picture deleted from program

Programmer Response: Probable user error. Correct picture. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0762I PICTURE CHARACTER E OR K IS NOT PRECEDED BY A DIGIT POSITION CHARACTER. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: References to picture deleted

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0763I INVALID CHARACTER nnnn IN EXPONENT FIELD OF PICTURE yyyy IN STATEMENT NUMBER xxx.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phase.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0764I ONE OR MORE FIXED BINARY ITEMS OF PRECISION 15 OR LESS HAVE BEEN GIVEN HALFWORD STORAGE. THEY ARE FLAGGED '*********' IN THE XREF/ATR LIST.

S IEM0769I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx AS EXPANDED IS TOO LONG AND HAS BEEN DELETED.

Programmer Response: Probable user error. Simplify by splitting into two or more statements and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0770I COMPILER ERROR IN INPUT TO PHASE GA IN STATEMENT NUMBER xxx

Explanation: The compiler has encountered meaningless input to phase GA.

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0771I IMPLEMENTATION RESTRICTION. NESTING OF FORMAT LISTS IN STATEMENT NUMBER xxx EXCEEDS 20. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0778I AN INTERMEDIATE VARIABLE HAS BEEN CREATED IN READ INTO STATEMENT NUMBER xxx. THIS STATEMENT SPECIFIES FILE zzzz, WHICH HAS BEEN DECLARED WITH THE ENV(COBOL) ATTRIBUTE AND THE EVENT OPTION. THE EVENT OPTION HAS BEEN DELETED.

Explanation: The intermediate variable has been created because there is a difference between the PL/I mapping and the COBOL mapping of the READ INTO variable. The READ INTO statement has been expanded into:

```
READ INTO (Intermediate
  variable);
Variable = Intermediate
  variable;
```

The READ statement must have been completed before the assignment takes place. The EVENT option has been deleted to ensure that the READ statement is complete before processing continues.

System Action: Delete the EVENT option and continue

Programmer Response: Probable user error. Check the use of the EVENT option or of the COBOL file. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0779I AN INTERMEDIATE VARIABLE HAS BEEN CREATED IN WRITE/REWRITE FROM STATEMENT NUMBER xxx. THIS STATEMENT SPECIFIES FILE zzzz, WHICH HAS BEEN DECLARED WITH THE ENV(COBOL) ATTRIBUTE AND THE EVENT OPTION. THE EVENT OPTION HAS BEEN DELETED.

Explanation: The intermediate variable has been created because there is a difference between the PL/I mapping and the COBOL mapping of the WRITE/REWRITE FROM variable. The WRITE/REWRITE FROM statement has been expanded to:

```
Intermediate
  variable = variable;
WRITE/REWRITE FROM
  (Intermediate variable);
```

The WRITE/REWRITE statement must have been completed before the intermediate variable can be deleted. The EVENT option

has been deleted to ensure that the WRITE/REWRITE statement is complete before processing continues.

System Action: Delete the EVENT option and continue

Programmer Response: Probable user error. Check the use of the EVENT option or of the COBOL file. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0780I THE USE OF COBOL FILE zzzz IN LOCATE STATEMENT NUMBER xxx MAY LEAD TO ERRORS WHEN THE RECORD IS PROCESSED.

Explanation: The COBOL structure-mapping is not necessarily the same as the PL/I structure-mapping.

System Action: The statement is deleted

Programmer Response: Probable user error. Either the LOCATE statement must be replaced by a WRITE FROM statement, or a non-COBOL file must be used. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0781I THE USE OF COBOL FILE zzzz IN READ SET STATEMENT NUMBER xxx MAY LEAD TO ERRORS WHEN THE RECORD IS PROCESSED.

Explanation: The COBOL structure-mapping is not necessarily the same as the PL/I structure-mapping.

System Action: The statement is deleted

Programmer Response: Probable user error. Either the READ SET statement must be replaced by a READ INTO statement, or a non-COBOL file must be used. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0782I THE ATTRIBUTES OF zzzz USED IN RECORD I/O STATEMENT NUMBER XXX ARE NOT PERMITTED WHEN A COBOL FILE IS USED.

Explanation: The attributes referred to do not exist in COBOL

System Action: The statement is deleted

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0784I INVALID ARGUMENT LIST FOR ALLOCATION FUNCTION IN STATEMENT NUMBER xxx HAS BEEN TRUNCATED OR DELETED.

Explanation: Only a single argument can be given in the ALLOCATION function, and it must be one of the following:

1. a major structure
2. an unsubscripted array or scalar variable, not in a structure

It must also be of nonbased CONTROLLED storage class.

System Action: If the argument list begins with a valid operand, that operand is used as the argument; otherwise, the argument list is deleted.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0786I NAME, NOT VALUE, OF FUNCTION zzzz PASSED AS ARGUMENT IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0787I INCORRECT NUMBER OF ARGUMENTS FOR FUNCTION OR ROUTINE zzzz IN STATEMENTS yyyy.

Explanation: Number of arguments differs from the ENTRY declaration.

System Action: Arguments are matched as far as possible. zzzz is invoked using all the arguments.

Programmer Response: Probable user error. Ignore this message if zzzz is a non-PL/I routine that can accept a variable number of arguments. Otherwise correct the program. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0791I NUMBER OF ARGUMENTS FOR FUNCTION OR SUBROUTINE zzzz IN STATEMENTS yyyy IS INCONSISTENT WITH NUMBER USED ELSEWHERE.

Explanation: The number of arguments for zzzz has not been explicitly declared. 'ELSEWHERE' refers either to the PROCEDURE or ENTRY statement for zzzz, or to a previous invocation of the function.

System Action: zzzz is invoked using all the arguments.

Programmer Response: Probable user error. Ignore this message if zzzz is a non-PL/I routine that can accept a variable number of arguments. Otherwise correct the program. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0792I IN STATEMENT NUMBER xxx IT IS IMPOSSIBLE TO CONVERT FROM THE ATTRIBUTES OF ARGUMENT NUMBER nnnn TO THOSE OF THE CORRESPONDING PARAMETER IN ENTRY zzzz. THE PARAMETER DESCRIPTION IS IGNORED.

Explanation: Self explanatory. Examples of circumstances under which the error message is generated are label arguments to data item parameters, array arguments to scalar parameters.

System Action: The parameter description is ignored. If the parameter description is correct, this will give rise to totally incorrect execution.

Programmer Response: Probable user error. Correct the parameter description or the argument so that at least conversion is possible. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0793I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS A SCALAR AND THE CORRESPONDING PARAMETER IS A STRUCTURE.

System Action: A temporary structure of the same type as the parameter description is created and the argument is assigned to each base element, converting where necessary.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0794I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz CONTAINS A SUBSCRIPTED VARIABLE WITH THE WRONG NUMBER OF SUBSCRIPTS. THE STATEMENT HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0795I DEFERRED FEATURE. IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz CONTAINS A

CROSS-SECTION OF AN ARRAY OF STRUCTURES. STATEMENT DELETED.

Explanation: The usage referred to is not supported by this version of the compiler. For details, refer to Appendix J of this publication.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0796I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS A GENERIC ENTRY NAME AND THERE IS NO CORRESPONDING ENTRY DESCRIPTION. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0797I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS A BUILT-IN FUNCTION WHICH MAY NOT BE PASSED AS AN ARGUMENT. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0798I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS NOT PERMISSIBLE. THE STATEMENT HAS BEEN DELETED.

Explanation: The message is generated, for example, when scalar arguments are given for array built-in functions. For the ADDR function the message could indicate that the function cannot return a valid result because the argument does not satisfy the requirements of contiguous storage. This could occur, for example, if the argument was a cross-section of an array.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0799I IN STATEMENT NUMBER xxx A DUMMY ARGUMENT HAS BEEN CREATED FOR ARGUMENT NUMBER nnnn OF ENTRY zzzz . THIS ARGUMENT APPEARS IN A SETS LIST.

System Action: The value assigned to the temporary argument during the execution of the procedure is lost on return from the procedure.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0800I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn IN ENTRY zzzz IS A SCALAR AND THE CORRESPONDING PARAMETER IS AN ARRAY.

System Action: A temporary array with the attributes of the entry description is created and the scalar is assigned to each element of the array, converting the type if necessary.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0801I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn IN ENTRY zzzz IS SCALAR CORRESPONDING TO AN ARRAY PARAMETER WITH * BOUNDS. THE STATEMENT HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program

and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0802I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz DOES NOT MATCH THE PARAMETER. A DUMMY ARGUMENT HAS BEEN CREATED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0803I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx CONTAINS TOO MANY NESTED FUNCTION REFERENCES. LIMIT EXCEEDED AT ARGUMENT NUMBER nnnn OF ENTRY zzzz

System Action: Compilation terminated

Programmer Response: Probable user error. Reduce depth of function call nesting. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM0804I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN DELETED.

System Action: Compilation terminated.

Programmer Response: Probable user error. Reduce statement size. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0805I DEFERRED FEATURE. IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS AN EVENT. STATEMENT DELETED.

Explanation: A language feature has been used that is not supported by this version of the compiler. For details, refer to Appendix J of this publication.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0806I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS NOT CONTROLLED BUT THE CORRESPONDING PARAMETER IS.

System Action: An execution error will occur on entry to the called procedure

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0807I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF BUILT-IN FUNCTION zzzz IS AN ENTRY NAME. THE STATEMENT HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0808I IN STATEMENT NUMBER xxx ARGUMENT NUMBER yyy WILL CAUSE A DUMMY ARGUMENT TO BE PASSED TO ENTRY yyy IN ANOTHER TASK.

Explanation: Dummy arguments passed to a subtask will cause temporary storage to be obtained which cannot be freed when the subtask is detached. This may result in the excessive use of storage.

Programmer Response: Probable user error. Correct the source program so that no dummy arguments are created for arguments passed to a subtask. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0816I COMPILER ERROR. INVALID END OF STATEMENT NUMBER xxx

Explanation: Compiler error in scan of input text

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM0817I COMPILER ERROR IN LABEL CHAIN FOR STATEMENT NUMBER xxx

Explanation: Compiler error in scanning labels of a statement

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

Note that this error can be avoided by using only one label on the statement.

T IEM0818I COMPILER ERROR IN DICTIONARY ENTRY FOR STATEMENT NUMBER xxx

Explanation: Compiler error in scanning source text

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM0819I COMPILER ERROR IN CHECK/NOCHECK LIST ENTRY FOR STATEMENT NUMBER xxx

Explanation: Compiler error in CHECK or NOCHECK list dictionary entry

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM0820I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx TOO LONG.

Explanation: Statement length exceeds text-block size.

System Action: Compilation is terminated.

Programmer Response: Probable user error. Subdivide statement and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM0821I INVALID ITEM zzzz IGNORED IN CHECK LIST IN STATEMENT NUMBER xxx

Explanation: Valid items in CHECK lists are: statement labels, entry labels, and scalar, array, structure, or label variables. Subscripted variable names, or data having the DEFINED attribute, are not allowed.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM0823I IMPLEMENTATION RESTRICTION. NO ROOM FOR zzzz IN CHECK TABLE STATEMENT NUMBER xxx

Explanation: The CHECK list table has overflowed

System Action: The item mentioned is ignored

Programmer Response: Probable user error. Do not CHECK so many items. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0824I IMPLEMENTATION RESTRICTION. TOO MANY CHECKED ITEMS WITHIN STATEMENT NUMBER xxx

Explanation: A stack used to trace nested IF statements has overflowed

System Action: Compilation is terminated

Programmer Response: Probable user error. Rephrase IF statements, or do not CHECK so many items. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM0825I COMPILER ERROR IN READ DATA STATEMENT NUMBER xxx

Explanation: Compiler error in processing GET or READ DATA statement

System Action: Compilation is terminated

Programmer Response: Avoid the error by supplying an explicit DATA list. Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages).
- Have the associated job stream and source program listing available.

W IEM0826I IMPLEMENTATION RESTRICTION. CHECK WILL NOT BE RAISED FOR zzzz IN STATEMENT NUMBER xxx BECAUSE OF EVENT OPTION

Explanation: The compiler does not raise the CHECK condition for variables when they are changed in statements containing an EVENT option.

Programmer Response: Probable user error. Correct program

and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0832I IN THE EXPANSION OF BY NAME ASSIGNMENT STATEMENT NUMBER xxx A SET OF MATCHING ELEMENTS HAS BEEN FOUND BUT THEY ARE NOT ALL BASE ELEMENTS. THE STATEMENT HAS BEEN DELETED.

Explanation: For a valid component scalar assignment to result from a BY NAME structure assignment, it is necessary that all the scalar names derived from original structure name operands have identical qualification relative to the structure name originally specified. e.g: DCL 1S, 2T, 2U, 3V; DCL 1W, 2T, 2U; W=S; gives rise to the component assignments W.T=S.T; W.U=S.U; the second of which is invalid.

System Action: The BY NAME assignment statement is deleted

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - rules for expansion of structure assignment BY NAME - and correct the error. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0833I THE EXPANSION OF BY NAME ASSIGNMENT STATEMENT NUMBER xxx HAS RESULTED IN NO COMPONENT ASSIGNMENTS.

System Action: The statement is treated as a null statement

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0834I THE ASSIGNED OPERAND IN BY NAME ASSIGNMENT STATEMENT NUMBER xxx IS NOT A STRUCTURE OR AN ARRAY OF STRUCTURES. STATEMENT DELETED.

Explanation: In BY NAME assignment, the operand to the left of the equals sign must be a structure or an array of structures.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0835I ALL OPERANDS LEFT OF EQUAL SYMBOL IN MULTIPLE STRUCTURE ASSIGNMENT STATEMENT NUMBER xxx ARE NOT STRUCTURES. STATEMENT DELETED

Explanation: In multiple structure assignment, all the operands being assigned to must be structures.

System Action: Replaces statement by a null statement and continues

Programmer Response: Probable user error. Break statement up into a series of separate statements. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0836I ILLEGAL ARRAY REFERENCE IN STRUCTURE ASSIGNMENT OR EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: In PL/I, arrays of scalars are invalid operands in structure, or array of structure, expressions.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0837I COMPILER ERROR IN INPUT TO PHASE IEMHF.

Explanation: Meaningless input. This message is also produced if an error is found in a DECLARE statement. In that case, a second message, of severity level severe, is issued giving details of the error.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM0838I EXPRESSION RIGHT OF EQUAL SYMBOL IN BY NAME ASSIGNMENT STATEMENT NUMBER xxx CONTAINS NO STRUCTURES. BY NAME OPTION DELETED.

Explanation: In an assignment statement having the BY NAME option, the expression to the right of the equal symbol should contain at least one structure or array of structures.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0848I IMPLEMENTATION RESTRICTION. THE EXPANSION OF STRUCTURE EXPRESSIONS IN STATEMENT NUMBER xxx HAS CAUSED A TABLE INTERNAL TO THE COMPILER TO OVERFLOW. STATEMENT DELETED.

Explanation: The nesting of structure expressions in argument lists is too deep.

Programmer Response: Probable user error. Decrease the nesting. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0849I AN EXPRESSION OR ASSIGNMENT IN STATEMENT NUMBER xxx EITHER CONTAINS SEPARATE STRUCTURES WITH DIFFERENT STRUCTURING, OR CONTAINS BOTH A STRUCTURE AND AN ARRAY OF STRUCTURES. THE STATEMENT HAS BEEN DELETED.

Explanation: PL/I does not allow separate structures with different structuring within the same expression or assignment. The (F) compiler does not support reference to both a structure and an array of structures within the same expression or assignment.

Programmer Response: Probable user error. Correct the source code. The two restrictions quoted above are detailed in the PL/I (F) Language Reference Manual and in Appendix J of this publication, respectively. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0850I THE BOUNDS OF THE BASE ARRAYS OF THE STRUCTURE OPERANDS OF THE STRUCTURE EXPRESSION OR ASSIGNMENT IN STATEMENT NUMBER xxx ARE NOT THE SAME. THE STATEMENT HAS BEEN DELETED.

Explanation: In a structure assignment or expression, all structure operands must have the same number of contained elements at the next level. Corresponding sets of contained elements must all be arrays of structures, structures, arrays, or scalars. The arrays must have the same dimensionality and bounds.

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - the expansion of array and structure expressions and assignments - and correct the program. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0851I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN DELETED.

Explanation: The expansion of structure expressions or assignments has given rise to a statement which exceeds one text block in length

Programmer Response: Probable user error. Decrease statement size. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0852I A SUBSCRIPTED REFERENCE TO AN ARRAY OF STRUCTURES IN STATEMENT NUMBER xxx HAS THE WRONG NUMBER OF SUBSCRIPTS. THE STATEMENT HAS BEEN DELETED.

Explanation: Subscripted references to arrays must have subscript expressions equal in number to the dimensionality of the array

Programmer Response: Probable user error. Correct subscripted reference. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0853I DEFERRED FEATURE. A STRUCTURE ASSIGNMENT OR EXPRESSION IN STATEMENT NUMBER xxx INVOLVES CROSS SECTIONS OF ARRAYS. STATEMENT DELETED.

Explanation: This version of the compiler does not support reference to cross sections of arrays of structures.

Programmer Response: Probable user error. Expand the statement in DO loops replacing '*'s in subscripts by the appropriate DO control variable. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0864I IMPLEMENTATION RESTRICTION. NESTING OF ARRAY ASSIGNMENTS OR I/O LISTS TOO DEEP. STATEMENT NUMBER xxx DELETED.

Explanation: Nesting of functions, combined with size of arrays involved, is too great

Programmer Response: Probable user error. Simplify by splitting into two or more statements. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0865I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN DELETED.

Explanation: Nesting of functions with array arguments involves a large expansion of text

Programmer Response: Probable user error. Simplify by splitting into two or more statements. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0866I NEITHER MULTIPLE ASSIGNMENT COMMA NOR ASSIGNMENT MARKER FOUND IN CORRECT POSITION. STATEMENT NUMBER xxx DELETED.

Explanation: An expression occurring on the left-hand side of an assignment must be contained within parentheses.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0867I NUMBER OF * SUBSCRIPTS SPECIFIED FOR zzzz IS NOT THE DIMENSIONALITY OF THE LEFTMOST ARRAY. STATEMENT NUMBER xxx DELETED

Explanation: Array references in expressions must have the same dimensionality

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0868I BOUNDS OF ARRAY zzzz ARE NOT SAME AS FOR LEFTMOST ARRAY IN ASSIGNMENT OR EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: Array references in expressions must have the same bounds

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0869I DIMENSIONS OF ARRAY zzzz ARE NOT THE SAME AS FOR LEFTMOST ARRAY IN ASSIGNMENT OR I/O EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: Array references in expressions must have the same dimensionality

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0870I ARGUMENT OF PSEUDO-VARIABLE INVALID. STATEMENT NUMBER xxx IS DELETED.

Explanation: Arguments of pseudo-variables must be variables which are not expressions

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0871I SCALAR zzzz ON LEFT OF EQUAL SYMBOL IN ARRAY ASSIGNMENT. STATEMENT NUMBER xxx DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0872I NUMBER OF SUBSCRIPTS SPECIFIED FOR LEFTMOST OPERAND zzzz IS NOT SAME AS DIMENSIONALITY. STATEMENT NUMBER xxx DELETED.

Explanation: The number of subscripts specified must be the same as the number of dimensions of the array

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0873I ARGUMENTS OF PSEUDO-VARIABLE COMPLEX INCORRECT. STATEMENT NUMBER xxx DELETED.

Explanation: Only one argument given for COMPLEX, or expression illegally used as argument for pseudo-variable.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0874I SECOND ARGUMENT OF PSEUDO-VARIABLE COMPLEX OR FIRST ARGUMENT OF REAL, IMAG, OR UNSPEC EITHER IS NOT FOLLOWED BY A RIGHT PARENTHESIS OR IS AN EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: Too many arguments given for this pseudo-variable, or expression used as argument.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0875I ONSOURCE OR ONCHAR APPEARS IN A DIMENSIONED ARRAY ASSIGNMENT. STATEMENT NUMBER xxx DELETED.

Explanation: ONSOURCE and ONCHAR may only appear in scalar assignments

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0876I ARGUMENTS OF PSEUDO-VARIABLE SUBSTR INCORRECT. STATEMENT NUMBER xxx DELETED.

Explanation: Only one argument given for SUBSTR, or expression illegally used as argument for pseudo-variable.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0877I UNSUBSCRIPTED ARRAY zzzz IN SCALAR ASSIGNMENT. STATEMENT NUMBER xxx DELETED.

Explanation: Only scalars can be assigned to scalars

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0878I STRUCTURE zzzz FOUND IN ARRAY OR SCALAR EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: Structures may only be assigned to structures

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0879I PSEUDO-VARIABLE COMPLEX, REAL, IMAG, UNSPEC, COMPLETION, OR SUBSTR LACKS ARGUMENTS. STATEMENT NUMBER xxx DELETED.

Explanation: Pseudo-variables COMPLEX, REAL, IMAG, UNSPEC, COMPLETION, and SUBSTR require arguments

Programmer Response: Probable user error. Check whether the variable was intended as a pseudo-variable or whether it should have been declared otherwise. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0880I NUMBER OF SUBSCRIPTS SPECIFIED FOR zzzz IS NOT SAME AS DIMENSIONALITY. STATEMENT NUMBER xxx DELETED.

Explanation: The number of subscripts specified must be the same as the number of dimensions of the array.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM0881I NUMBER OF DIMENSIONS IN REFERENCE TO zzzz IS NOT SAME AS THAT OF EXPRESSION OR ASSIGNMENT. STATEMENT NUMBER xxx DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0882I COMPILER ERROR. INVALID INPUT TO PHASE HK AT STATEMENT NUMBER xxx

Explanation: Illegal text has been encountered

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0883I DEFERRED FEATURE. STRUCTURE zzzz PASSED AS ARGUMENT TO THE TRANSLATE OR VERIFY FUNCTION. STATEMENT NUMBER xxx DELETED.

Programmer Response: Probable user error. Remove structure from statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0896I IMPLEMENTATION RESTRICTION. TOO MANY LEVELS OF ISUB NESTING IN STATEMENT NUMBER xxx

Explanation: Stack has overflowed scratch core. The maximum number of levels of nesting possible depends on the dimensionality of the arrays involved.

System Action: Compilation terminated

Programmer Response: Probable user error. Reduce the number of levels of nesting in the statement. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM0897I ISUB DEFINED OPERAND zzzz HAS NOT BEEN DECLARED AS AN ARRAY. ISUBS IN STATEMENT NUMBER xxx DELETED.

Programmer Response: Probable user error. Declare defined item with dimension attribute. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0898I NO SUBSCRIPTS AFTER ISUB-DEFINED ITEM zzzz IN STATEMENT NUMBER xxx.

Explanation: This is a compiler error

System Action: Terminates compilation

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM0899I MULTIPLIER IN ISUB DEFINING LIST FOR zzzz IN STATEMENT NUMBER xxx IS NOT A SCALAR EXPRESSION.

Explanation: A comma has been found within the iSUB multiplier expression

System Action: Remainder of expression, after comma, ignored

Programmer Response: Probable user error. Rewrite expression. If the problem recurs, do the following before

calling IBM for programming support:

- Have the source program listing available.

E IEM0900I * USED AS SUBSCRIPT FOR ISUB DEFINED ITEM zzzz IN STATEMENT NUMBER xxx. ZERO SUBSTITUTED.

Programmer Response: Probable user error. Rewrite without *. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0901I ISUB NUMBER IN DEFINING LIST FOR zzzz IN STATEMENT NUMBER xxx IS TOO GREAT. MAXIMUM NUMBER USED.

System Action: The iSUB number is replaced by the number of dimensions of the defined array.

Programmer Response: Probable user error. Rewrite defining DECLARE statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM0902I WRONG NUMBER OF SUBSCRIPTS FOR ISUB DEFINED ITEM zzzz IN STATEMENT NUMBER xxx. SUBSCRIPTS IGNORED OR ZERO SUPPLIED.

Programmer Response: Probable user error. Rewrite with correct number of subscripts. The error may be in the reference to the defined item or in the defining DECLARE statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM0903I COMPILER ERROR. ERROR DETECTED IN DEFINING ISUB LIST FOR zzzz IN STATEMENT NUMBER xxx

Explanation: Compiler error. Either (a) SUB not found where expected, or (b) SUB0 found without a multiplier expression.

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

S IEM0906I STATEMENT DELIMITER FOUND WITHIN SUBSCRIPT LIST FOR zzzz IN STATEMENT NUMBER xxx

Explanation: The subscript scan routine has found a statement marker

System Action: The present statement is dropped and the new one processed. Compilation will not be completed.

Programmer Response: Check the source text. This is probably a compiler error; do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,HP)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

S IEM0907I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN TRUNCATED.

Explanation: Statement length exceeds text block size

System Action: Statement is truncated. Compilation will not be completed.

Programmer Response: Probable user error. Simplify statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1024I ILLEGAL USE OF zzzz IN STATEMENT NUMBER xxx. A FIXED BINARY ZERO CONSTANT IS SUBSTITUTED.

Explanation: A non-scalar identifier has been specified in a context that requires a scalar identifier.

System Action: Replaces illegal identifier with arithmetic constant zero

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1025I IDENTIFIER zzzz ILLEGALLY USED AS SUBSCRIPT IN STATEMENT NUMBER xxx

Explanation: A subscript has been used which is not a scalar, a scalar expression, or a constant

System Action: Replaces illegal subscript with arithmetic constant zero

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1026I STATEMENT NUMBER xxx IS AN UNLABELED FORMAT STATEMENT

Explanation: A FORMAT statement should have a label

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1027I THE SUBSCRIPTED STRUCTURE ITEM zzzz IS ILLEGALLY USED IN STATEMENT NUMBER xxx

Explanation: The indicated structure item is used in a statement other than an assignment statement or an I/O data list.

System Action: Compilation is terminated

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1028I COMPILER ERROR IN STATEMENT NUMBER xxx. ILLEGAL INPUT TEXT FOR PHASE IA.

System Action: Terminates compilation

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1029I THE APPEARANCE OF THE ARRAY CROSS-SECTION IN STATEMENT NUMBER xxx IS NOT SUPPORTED BY THIS VERSION OF THE COMPILER.

Explanation: A feature has been used that is not supported by this version of the compiler. For details, refer to Appendix J or to IBM System/360 Operating System: PL/I (F) Language Reference Manual.

System Action: Terminates compilation

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1030I IMPLEMENTATION RESTRICTION. TOO MANY DUMMY ARGUMENTS ARE BEING PASSED IN STATEMENT NUMBER xxx. A MAXIMUM OF 64 DUMMY ARGUMENTS MAY BE PASSED IN EACH INVOCATION

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1040I DEFERRED FEATURE. STRUCTURE ARGUMENT IS BEING PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx.

Explanation: In this version of the compiler, structures may not be passed as arguments to built-in functions.

System Action: Terminates compilation

Programmer Response: Probable user error. Rewrite program, avoiding unsupported feature. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1051I DEFERRED FEATURE. STRUCTURE ARGUMENT IS BEING PASSED TO PSEUDO-VARIABLE zzzz IN STATEMENT NUMBER xxx.

Explanation: In this version of the compiler, structures may not be passed as arguments to pseudo-variables.

System Action: Terminates compilation

Programmer Response: Probable user error. Rewrite program, avoiding unsupported feature. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1056I INVALID ARGUMENT IS BEING PASSED TO ENTRY NAME zzzz IN STATEMENT NUMBER xxx.

System Action: Terminates compilation

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1057I DECIMAL INTEGER CONSTANT IS NOT BEING PASSED, AS REQUIRED, TO FUNCTION zzzz IN STATEMENT NUMBER xxx.

Explanation: Argument to built-in function is not a decimal integer as expected.

System Action: Terminates compilation

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1058I ARRAY OR STRUCTURE ARGUMENT IS NOT BEING PASSED, AS REQUIRED, TO FUNCTION zzzz IN STATEMENT NUMBER xxx.

Explanation: Argument to built-in function is not an array or a structure, as expected.

System Action: Terminates compilation.

Programmer Response: Probable user error. Correct statement. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1059I FIRST ARGUMENT BEING PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx SHOULD BE AN ARRAY.

Explanation: Argument to built-in function is not an array as expected

System Action: Terminates compilation

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1060I TOO MANY ARGUMENTS ARE BEING PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx.

Explanation: Too many arguments are being passed to a built-in function

System Action: Terminates compilation

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1061I TOO FEW ARGUMENTS ARE BEING PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx.

Explanation: Too few arguments are being passed to a built-in function

System Action: Terminates compilation

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job

stream and source program listing available.

T IEM1062I COMPILER ERROR. CORRECT GENERIC SELECTION FOR FUNCTION zzzz IN STATEMENT NUMBER xxx HAS NOT BEEN ACHIEVED.

Explanation: Compiler, although being given a legal argument to a generic built-in function, is unable to make the selection.

System Action: Function result is set to zero and compilation terminated.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1063I COMPILER ERROR. UNEXPECTED SITUATION HAS ARISEN IN THE SCANNING OF THE ARGUMENTS PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx

Explanation: Compiler is unable to correctly scan an argument list

System Action: Function result is set to zero

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1064I COMPILER ERROR. THE GENERIC FAMILIES ASSOCIATED WITH ENTRY NAME zzzz HAVE BEEN INCORRECTLY FORMED IN THE DICTIONARY.

Explanation: The dictionary entry for one or more of the generic families is not a recognizable entry type.

System Action: Terminates compilation

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1065I NO GENERIC SELECTION POSSIBLE FOR THE ENTRY NAME zzzz IN STATEMENT NUMBER xxx.

Explanation: Incorrect use of the GENERIC attribute resulting in no selection being possible

System Action: Terminates compilation

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1066I MORE THAN ONE GENERIC SELECTION IS POSSIBLE FOR THE ENTRY NAME zzzz IN STATEMENT NUMBER xxx.

Explanation: Incorrect use of the GENERIC attribute resulting in more than one selection being possible

System Action: Terminates compilation

Programmer Response: Probable user error. If the problem recurs, do the following before

calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1067I PSEUDO-VARIABLE zzzz APPEARS IN STATEMENT NUMBER xxx WITH AN ILLEGAL ARGUMENT.

Explanation: Argument to pseudo-variable cannot be converted to a legal type; or, structure argument being used with pseudo-variable.

System Action: Terminates compilation

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1068I AN ARRAY IS BEING PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx. THIS PRODUCES AN ARRAY EXPRESSION WHICH IS INVALID IN THIS CONTEXT.

System Action: Terminates compilation

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM1070I IMPLEMENTATION RESTRICTION. AN ARGUMENT OF A BUILT-IN FUNCTION USED IN STATEMENT NUMBER xxx HAS BEEN TRUNCATED TO 32,767.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1071I PSEUDO-VARIABLE zzzz APPEARS IN STATEMENT NUMBER xxx WITH TOO MANY ARGUMENTS.

System Action: Terminates compilation

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1072I PSEUDO-VARIABLE zzzz APPEARS IN STATEMENT NUMBER xxx WITH TOO FEW ARGUMENTS.

System Action: Terminates compilation

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1073I COMPILER ERROR. CORRECT GENERIC SELECTION FOR PSEUDO-VARIABLE zzzz IN STATEMENT NUMBER xxx HAS NOT BEEN ACHIEVED.

Explanation: Compiler error. Although being given a legal argument to a generic pseudo-variable, is unable to make the selection.

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1074I COMPILER ERROR. UNEXPECTED SITUATION HAS ARISEN IN THE SCANNING OF THE ARGUMENTS PASSED TO PSEUDO-VARIABLE zzzz IN STATEMENT NUMBER xxx

Explanation: Unable to correctly scan an argument list of a pseudo-variable

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

W IEM1075I THE ARGUMENT zzzz OF THE STRING PSEUDO-VARIABLE IN STATEMENT NUMBER xxx CONTAINS A PICTURED ELEMENT. THIS IS NOT CHECKED FOR VALIDITY ON ASSIGNMENT.

Explanation: Invalid data in pictured element may cause subsequent errors.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1076I COMPILER ERROR IN PHASE JD

System Action: Compilation terminates

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1078I IMPLEMENTATION RESTRICTION. THE NUMBER OF FAMILY MEMBERS AND ARGUMENTS ASSOCIATED WITH THE GENERIC ENTRY NAME yyyy EXCEEDS THE LIMITATION IMPOSED.

Explanation: There is an implementation restriction on the number of family members and arguments associated with GENERIC entry names. For details, refer to Appendix J of this publication.

System Action: Compilation terminated

Programmer Response: Probable user error. Divide the generic family into two or more generic families. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

S IEM1082I STATEMENT NUMBER xxx CONTAINS AN INVALID USE OF AREA OR POINTER DATA. PART OR ALL OF THE STATEMENT HAS BEEN DELETED.

Explanation: The statement contains an operation that:

1. is not permitted for AREA or POINTER data, or
2. can only be used with AREA or POINTER data but such data is not the data specified for the operation.

System Action: Deletes the statement or clause responsible for the error.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1088I THE SIZE OF AGGREGATE zzzz IS GREATER THAN 8,388,607 BYTES. STORAGE ALLOCATION WILL BE UNSUCCESSFUL.

Explanation: The message is generated when an array or structure size exceeds $2^{23}-1$

System Action: Array or structure mapping for the item is terminated, but the compilation continues. Execution of object decks containing references to the item will give incorrect results.

Programmer Response: Probable user error. Check source code. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1089I THE RELATIVE VIRTUAL ORIGIN OF AGGREGATE zzzz IS LESS THAN -8,388,608 BYTES. STORAGE HAS NOT BEEN ALLOCATED.

Explanation: The low bounds of the arrays in the aggregate are too high.

System Action: Compilation is terminated.

Programmer Response: Probable user error. Reduce the size of the aggregate, or reduce the value of the low bounds in the aggregate. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM1090I THE STRUCTURE zzzz DECLARED IN STATEMENT NUMBER xxx CONTAINS VARYING STRINGS AND MAY APPEAR IN A RECORD I/O STATEMENT

Explanation: VARYING strings in structures are not permitted in RECORD I/O statements.

System Action: The RECORD I/O statement is processed but the record will contain erroneous information.

Programmer Response: Probable user error. Correct the source code. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1092I THE TASKS, EVENTS OR LABELS CONTAINED IN STRUCTURE zzzz DECLARED IN STATEMENT NUMBER xxx MAY LOSE THEIR VALIDITY IF USED IN A RECORD I/O STATEMENT.

Explanation: The TASK, EVENT, or LABEL variable may lose its validity in transmission.

Programmer Response: Probable user error. Correct the source code if necessary. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1104I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx INVOLVES DATA NOT ALLOWED FOR STRING CLASS OVERLAY DEFINING.

Explanation: The programmer's use of the DEFINED attribute contravenes the language rules concerned with the permitted data types and dimensionality of base and defined item.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - "The DEFINED Attribute" - and correct the error. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1105I THE DATA CHARACTERISTICS OF zzzz DECLARED IN STATEMENT NUMBER xxx DO NOT MATCH THOSE OF THE DEFINING BASE.

Explanation: For valid use of the DEFINED attribute, both the defined item and the base must be of the same defining class.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - "The DEFINED Attribute" - and correct the error. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1106I THE DIMENSIONALITY OF zzzz DECLARED IN STATEMENT NUMBER xxx IS NOT THE SAME AS THAT OF THE DEFINING BASE.

Explanation: With the exception of the case of string class defining, if either the base or the defined item are arrays, then both the base and the defined item must be arrays with the same dimensionality.

System Action: Compilation is aborted after examining other uses of the DEFINED attribute

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - "The DEFINED Attribute" - and correct the error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1107I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx ILLEGALLY INVOLVES VARYING STRINGS.

Explanation: In use of the DEFINED attribute, neither the base nor the defined item may involve strings declared VARYING.

System Action: Compilation is aborted after examining other uses of the DEFINED attribute.

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - "The DEFINED Attribute" - and correct the error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the

comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

E IEM1108I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx ILLEGALLY INVOLVES DATA AGGREGATES THAT ARE NOT UNALIGNED.

Explanation: In the case of string class overlay defining where either or both the base and the defined item are aggregates, then the aggregates must have the PACKED attribute.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - "The DEFINED Attribute" - and correct the error. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1110I THE DEFINING BASE OF zzzz DECLARED IN STATEMENT NUMBER xxx IS SHORTER THAN THE DEFINED ITEM.

Explanation: In the case of string class overlay defining, the defined item must occupy a subset of the base storage.

In the case of correspondence defining, the length of each defined element must not be greater than the length of each base element.

System Action: Compilation is aborted after examining other uses of the DEFINED attribute

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - "The DEFINED Attribute" - and correct the error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

E IEM1111I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx INVOLVES A STRUCTURE HAVING ELEMENTS NOT ALL OF THE SAME DEFINING CLASS.

Explanation: In the case of string class overlay defining where the defined item or the base is a structure, then all the elements of the structure must be data of the same string defining class.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - "The DEFINED Attribute" - and correct the error. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1112I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx ILLEGALLY INVOLVES THE POS ATTRIBUTE.

Explanation: The POSITION attribute may only be declared for data of the string class which is overlay defined

System Action: Compilation is terminated

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - "The DEFINED Attribute" - and correct the error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a

formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

E IEM1113I THE STRUCTURE DESCRIPTION OF zzzz DECLARED IN STATEMENT NUMBER xxx DOES NOT MATCH THAT OF THE DEFINING BASE.

Explanation: Where a structure or an array of structures is defined on a structure or an array of structures, and it is not string class overlay defining, then the two structure descriptions must be identical.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

Programmer Response: Probable user error. Refer to the PL/I (F) Language Reference Manual - "The DEFINED Attribute" - and correct the error. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1114I IF THE BASE OF zzzz DECLARED IN STATEMENT NUMBER xxx IS ALLOCATED WITH THE DECLARED EXTENTS, THE DEFINING WILL BE IN ERROR.

Explanation: In the case of string class overlay defining, the defined item must occupy a subset of the base storage. If the base is of CONTROLLED storage class, its extents are not finally resolved until execution time.

System Action: No further action

Programmer Response: Probable user error. Check that when the base is allocated it is of adequate size to accommodate the defined item. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1115I THE DEFINING BASE OF zzzz DECLARED IN STATEMENT NUMBER xxx IS AN ARRAY FORMAL PARAMETER. IF THE MATCHING ARGUMENT IS AN ELEMENT OF AN ARRAY OF STRUCTURES OR A CROSS SECTION OF AN ARRAY, THE DEFINING WILL BE IN ERROR.

Explanation: The base for string class overlay defining must occupy contiguous storage

System Action: Comments and continues

Programmer Response: Probable user error. Check validity of arguments. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1120I COMPILER ERROR. INVALID SIGN FOUND IN INITIAL VALUE LIST FOR zzzz IN STATEMENT NUMBER xxx. TREATED AS PLUS.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,JP)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

S IEM1121I COMPILER ERROR. INVALID MARKER FOUND IN INITIAL VALUE LIST FOR zzzz IN STATEMENT NUMBER xxx. INITIAL VALUE LIST TRUNCATED.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,JP)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job

stream and source program listing available.

S IEM1122I UNSUPPORTED FEATURE. AN EXPRESSION HAS BEEN USED TO INITIALIZE STATIC STRING zzzz IN STATEMENT NUMBER xxx. STRING INITIALIZED TO NULL.

Explanation: A complex expression has been used to initialize a STATIC string. This is a feature of PL/I not supported by this version of the compiler. See Appendix J of this publication for details.

System Action: The string is initialized to null.

Programmer Response: Probable user error. Amend source code. The restriction can be overcome by using an assignment statement instead of the INITIAL attribute. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1123I INITIAL VALUE FOR STATIC DATA ITEM zzzz IN nnnn IS NOT A CONSTANT. INITIALIZATION TERMINATED.

Programmer Response: Probable user error. Use a constant in the INITIAL string. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1125I ITERATION FACTOR USED IN INITIALIZATION OF STATIC ARRAY zzzz IN STATEMENT NUMBER xxx IS TOO LARGE. REPLACED BY ZERO.

Explanation: Iteration factors are converted by the compiler to REAL FIXED BINARY with a default precision of 15,0. The iteration factor referred to in the message has a value greater than 2^{15}, and therefore exceeds the default precision.

System Action: The iteration factor is replaced by zero

Programmer Response: Probable user error. Amend source code so that iteration factor does not exceed 2^{15}. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1200I COMPILER ERROR. ILLEGAL TRIPLE IN TEXT. CURRENT STATEMENT NUMBER xxx

Explanation: Phase KT is out of step in scanning text

System Action: Recovery impossible. Compilation is terminated.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1210I COMPILER ERROR NUMBER nnnn IN PHASE KE.

Explanation: Compiler error in dictionary or text scan.

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.
- Recompilation should also be tried without the OPT=2 option.

T IEM1211I COMPILER ERROR IN PHASE KE.

Explanation: Compiler error found in scan of dictionary.

System Action: Compilation terminated.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1220I COMPILER ERROR NUMBER nnnn IN PHASE KU IN STATEMENT NUMBER xxx.

Explanation: A compiler error has occurred in the DO loop control optimization phase.

System Action: Compilation is terminated.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.
- Recompilation should also be tried without the OPT=2 option.

T IEM1223I COMPILER ERROR. INVALID INPUT TYPE nnnn TO OPTIMIZING PHASE KO.

System Action: Compilation terminated.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.
- Recompilation should also be tried without the OPT=2 option.

T IEM1224I COMPILER ERROR NUMBER nnnn IN PHASE KA.

Explanation: An invalid request has been encountered by the table-handling routines in Phase KA.

System Action: Compilation is terminated.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1226I COMPILER ERROR IN PHASE KG IN OR NEAR STATEMENT NUMBER xxx

Explanation: An error has occurred while scanning text or tables.

System Action: Compilation terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.
- Recompilation should also be tried without the OPT=2 option.

T IEM1569I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE.

Explanation: The number of symbolic register names generated by the code generation section of the compiler has exceeded the maximum number allowed

System Action: Compilation is terminated

Programmer Response: Probable user error. Programmer should break down the compilation into smaller modules. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1570I COMPILER ERROR. INVALID TRIPLE FOLLOWING WHILE PRIME TRIPLE.

Explanation: Input to phase LG of compiler is erroneous. A WHILE' triple is not followed by CV' or compiler label.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1571I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE.

Explanation: No more core is available for the stack of nested DO statements (both in source language and those generated internally for array assignments etc.)

System Action: Compilation is terminated

Programmer Response: Probable user error. Simplify nesting so as to reduce number of levels. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM1572I ILLEGAL USE OF ARRAY OR STRUCTURE VARIABLE IN DO STATEMENT NUMBER xxx

Explanation: A non-scalar variable has been used as (1) the control variable, or (2) a control variable subscript, or (3) a loop limit or increment value.

System Action: Generates an error stop at execution time

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1574I INVALID LOOP CONTROL EXPRESSION OR CONTROL VARIABLE SUBSCRIPT IN STATEMENT NUMBER xxx REPLACED BY FIXED BINARY TEMPORARY.

Explanation: Either something other than an arithmetic or string datum has been used as a subscript in the control variable, or something other than an arithmetic or string datum, label variable, or label constant has been used in an initial value, TO or BY clause.

System Action: Ignore the erroneous expression and use a fixed binary temporary

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1575I DO LOOP CONTROL PSEUDO-VARIABLE IN STATEMENT NUMBER xxx HAS AN INVALID ARGUMENT. BINARY INTEGER TEMPORARY ASSUMED.

Explanation: An invalid argument, such as an expression or function, has been used in a pseudo-variable.

System Action: Assigns invalid argument to binary temporary, and uses the latter as argument.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1588I VARYING STRING HAS BEEN USED AS AN ARGUMENT TO ADDR FUNCTION IN STATEMENT NUMBER xxx

Explanation: The result of the ADDR function can only be assigned to a pointer qualifying a based variable. If the argument to the ADDR function is a VARYING string, the length of the data in the based variable may not be the length required in the program.

System Action: None

Programmer Response: Probable user error. Check this use of the ADDR function. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1599I A STATEMENT LABEL CONSTANT IS BEING PASSED AS AN ARGUMENT TO THE ADDR BUILT-IN FUNCTION IN STATEMENT NUMBER xxx

Explanation: The argument to the ADDR built-in function must be a variable.

Programmer Response: Probable user error. Correct source program. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1600I COMPILER ERROR. ILLEGAL ABSOLUTE REGISTER NUMBER. STATEMENT NUMBER xxx

Explanation: Compiler error. Fixed binary arithmetic uses an unassigned general register number greater than 15, or floating arithmetic uses a floating register greater than 6.

System Action: Compilation is terminated and error messages printed.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1601I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx REQUIRES MORE THAN 200 INTERMEDIATE RESULT DESCRIPTIONS.

Explanation: Compiler limitation. The temporary result stack, which holds 200 items, is full.

System Action: Compilation is terminated and error messages printed

Programmer Response: Probable user error. This error should only occur in very large statements. Divide the statement into two smaller statements. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM1602I COMPILER ERROR. INSUFFICIENT NUMBER OF TEMPORARY RESULT DESCRIPTIONS. STATEMENT NUMBER xxx

Explanation: Compiler error. A temporary result is required but the temporary result stack is empty. This can happen if the triples are out of order or if extra triples have been inserted.

System Action: Compilation is aborted and error messages printed

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM1603I COMPILER ERROR. COUNT OF FREE FLOATING REGISTERS IS WRONG. STATEMENT NUMBER xxx

Explanation: Compiler error in expression evaluation phase. Error in control blocks for floating registers.

System Action: Compilation is terminated and error messages printed

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM1604I COMPILER ERROR. SECOND OPERAND FOR RS OR SS INSTRUCTION IS IN A REGISTER. STATEMENT NUMBER xxx

Explanation: Compiler error in expression evaluation phase. Attempt to generate an RS or SS type pseudo-code instruction using a register as the 2nd operand.

System Action: Compilation is terminated and error messages printed.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

S IEM1605I IN STATEMENT NUMBER xxx FIXED DECIMAL VARIABLE CANNOT BE CORRECTLY CONVERTED TO BINARY DUE TO SIZE OF SCALE FACTOR.

Explanation: Error in source program. When a fixed decimal variable is corrected to fixed binary, the magnitude of its scale factor is multiplied by 3.31. If the original scale factor is >38 or <-38, then the fixed binary scale factor would be outside the range +127 to -128.

System Action: The fixed binary scale factor is set to +127 or -128. Processing continues.

Programmer Response: Probable user error. The data in the expression must be re-declared with more suitable scale factors. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1606I COMPILER ERROR. FUNCTION NOT FOLLOWED BY RESULT DESCRIPTION. STATEMENT NUMBER xxx

Explanation: Compiler error. A function is not followed by TMPD or LEFT triples giving the result type.

System Action: Compilation is terminated and error messages printed

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM1607I LABEL, EVENT, FILE, OR TASK ITEM zzzz IN STATEMENT NUMBER xxx IS USED IN AN EXPRESSION WHICH IS ILLEGAL.

Explanation: Error in source program. A label, event, file, or task datum cannot be used in an expression. Alternatively, this can be a compiler error when an unrecognizable dictionary entry is used in an expression.

System Action: Substitute a fixed binary (31,0) data item (if the illegal item occurs in an arithmetic expression) or a null bit string (if it occurs in a string expression). Processing is continued.

Programmer Response: Probable user error. If error in source program, correct it. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1608I LT, LE, GE, OR GT COMPARISON OPERATOR ILLEGALLY USED IN STATEMENT NUMBER xxx WITH COMPLEX OPERANDS. REPLACED WITH EQUALS OPERATOR.

Explanation: Error in source program. The only legal comparison between complex operands is '='.

System Action: The operator is replaced with '=' and processing continues

Programmer Response: Probable user error. Correct source program using either the ABS function or possibly the REAL and IMAG functions. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1609I COMPILER ERROR. ILLEGAL DICTIONARY REFERENCE X'00..' STATEMENT NUMBER xxx

Explanation: Compiler error. The symbolic dictionary reference is less than 256.

System Action: Compilation terminated and error messages printed

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1610I COMPILER ERROR IN PHASE LW AT STATEMENT NUMBER xxx. INSUFFICIENT NUMBER OF TEMPORARY RESULT DESCRIPTIONS.

Explanation: Compiler error. A temporary result is required but the temporary result stack is empty. This can happen if the triples are out of order or if extra triples have been inserted.

System Action: Compilation is terminated and error messages printed

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM1611I IMPLEMENTATION RESTRICTION. A STRING RESULT LONGER THAN 32767 IS PRODUCED BY CONCATENATE IN STATEMENT NUMBER xxx. STRING TRUNCATED TO LENGTH 32767.

Explanation: Maximum string length for this implementation is 32767. This may be exceeded during concatenation, because the length of the intermediate result is the sum of the operand lengths.

System Action: Compilation continues with string result length truncated to 32767

Programmer Response: Probable user error. Shorter strings must be used. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1612I IMPLEMENTATION RESTRICTION IN STATEMENT NUMBER xxx. INTERMEDIATE WORK SPACE IS OBTAINED MORE THAN 50 TIMES IN A STRING EXPRESSION. SOME WORK SPACE WILL NOT BE RELEASED UNTIL THE END OF THE BLOCK.

Explanation: The intermediate work space is required each time a function returns a string result or each time a library module is called

System Action: The first 50 areas of work space are released. The remainder may not be released until the end of the block. Compilation continues and execution is valid.

Programmer Response: Probable user error. Divide the string expression into several sub-expressions. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1613I ILLEGAL USE OF ARRAY OR STRUCTURE VARIABLE IN STATEMENT NUMBER xxx

Explanation: Illegal source program

System Action: Severe error message and object program branch. Compilation continues, assuming scalar of same type if array, or fixed binary (31,0) type if structure.

Programmer Response: Probable user error. Insert DO blocks for array, or break down structure into its components. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1614I IMPLEMENTATION RESTRICTION. A VARYING STRING RESULT LONGER THAN 32767 MAY BE PRODUCED BY CONCATENATE IN STATEMENT NUMBER xxx. STRING TRUNCATED TO LENGTH 32767.

Explanation: The sum of the maximum lengths of two strings in a concatenation operation exceeds the implementation restriction of 32767. Since one or both of the operands is a VARYING string, it is not known at compile-time whether the restriction will be exceeded at execution time.

System Action: Compilation continues with string result maximum length truncated to 32767.

Programmer Response: Probable user error. Shorter strings must be used if the sum of the execution-time current lengths will ever exceed 32767. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1615I SECOND ARGUMENT IN THE SUBSTR FUNCTION IN STATEMENT NUMBER xxx IS ZERO, WHICH IS INVALID. ZERO HAS BEEN REPLACED BY ONE.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1616I SECOND ARGUMENT IN THE SUBSTR PSEUDO-VARIABLE IN STATEMENT NUMBER xxx IS ZERO, WHICH IS INVALID. ZERO HAS BEEN REPLACED BY ONE.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1617I COMPILER ERROR. ILLEGAL RETURN FROM SCAN ROUTINE. STATEMENT NUMBER xxx

Explanation: An illegal return of control has been made by the SCAN routine which supports the code generation phases.

System Action: Compilation is terminated.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM1618I PSEUDO-VARIABLE IN STATEMENT NUMBER xxx INCORRECTLY SPECIFIED. REPLACED BY FIXED BINARY TEMPORARY.

Explanation: A pseudo-variable in the given source statement has been incorrectly specified, e.g. has an incorrect number of arguments.

System Action: Ignores the pseudo-variable and uses a fixed binary temporary instead.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1619I RIGHT HAND SIDE OF STATEMENT NUMBER xxx CANNOT BE ASSIGNED TO A PSEUDO-VARIABLE. ASSIGNMENT IGNORED.

Explanation: The expression on the right-hand side of the specified statement cannot be assigned to a pseudo-variable, i.e. it is not an arithmetic or string datum.

System Action: The assignment is deleted from the text.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1620I 'IMAG' IN STATEMENT NUMBER xxx HAS REAL ARGUMENT. REPLACED BY ASSIGNMENT TO TEMPORARY FIXED BINARY INTEGER.

Explanation: The pseudo-variable 'IMAG' is meaningful only if its argument is of type complex.

System Action: A fixed binary temporary target is provided for the assignment or input data list item and the pseudo-variable is ignored

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1621I ILLEGAL PSEUDO-VARIABLE ARGUMENT IN STATEMENT NUMBER xxx REPLACED BY BINARY TEMPORARY.

Explanation: A pseudo-variable in the specified statement has an illegal argument, i.e. one whose data type is not permissible in that context.

System Action: A temporary whose type is legal in the context is used to replace the erroneous argument and the latter is removed from the text

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1622I FIRST ARGUMENT OF PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx IS NOT A STRING VARIABLE. ARGUMENT HAS BEEN CONVERTED TO STRING TEMPORARY AND THE ASSIGNMENT MADE THERETO.

Explanation: SUBSTR pseudo-variable cannot have a first argument which is not a string variable.

System Action: Code is compiled to assign to a string temporary. The original argument remains unchanged.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1625I PSEUDO-VARIABLE REAL IN STATEMENT NUMBER xxx DOES NOT HAVE COMPLEX ARGUMENT. ARGUMENT HAS BEEN TREATED AS HAVING ZERO IMAGINARY PART.

System Action: Code is generated to perform assignment to the specified REAL argument

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1626I ILLEGAL NEGATIVE SECOND ARGUMENT IS BEING PASSED TO THE FUNCTION SUBSTR IN STATEMENT NUMBER xxx. AN EXECUTION ERROR WILL RESULT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1627I ILLEGAL NEGATIVE THIRD ARGUMENT IS BEING PASSED TO THE FUNCTION SUBSTR IN STATEMENT NUMBER xxx. AN EXECUTION ERROR WILL RESULT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1628I THE SUBSTRING SPECIFIED BY THE SECOND AND THIRD ARGUMENTS TO THE FUNCTION SUBSTR IN STATEMENT NUMBER xxx DOES NOT LIE WITHIN THE FIRST ARGUMENT. AN EXECUTION ERROR WILL RESULT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1629I THE SECOND ARGUMENT TO THE FUNCTION SUBSTR IN STATEMENT NUMBER xxx IS GREATER THAN THE LENGTH OF THE FIRST ARGUMENT. AN EXECUTION ERROR WILL RESULT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1630I COMPILER ERROR IN CEIL/FLOOR/TRUNC IN-LINE FUNCTION IN STATEMENT NUMBER xxx

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1631I COMPILER ERROR IN MOD IN-LINE FUNCTION IN STATEMENT NUMBER xxx

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

W IEM1632I THE INVOCATION OF THE ROUND FUNCTION IN STATEMENT NUMBER xxx WILL ALWAYS GIVE A ZERO RESULT.

Explanation: (p - q + r) is zero or negative, where p = precision, q = scale factor, and r = rounding position.

System Action: Result is set to zero

Programmer Response: Probable user error. Check scale and precision of the first argument in ROUND function. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1633I ILLEGAL NEGATIVE SECOND ARGUMENT IS BEING PASSED TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx. AN EXECUTION ERROR WILL RESULT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1634I ILLEGAL NEGATIVE THIRD ARGUMENT IS BEING PASSED TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx. AN EXECUTION ERROR WILL RESULT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1635I THE SUBSTRING SPECIFIED BY THE SECOND AND THIRD ARGUMENTS TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx DOES NOT LIE WITHIN THE STRING zzzz. AN EXECUTION ERROR WILL RESULT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1636I THE SECOND ARGUMENT TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx IS GREATER THAN THE LENGTH OF THE STRING zzzz. AN EXECUTION ERROR WILL RESULT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before

calling IBM for programming support:

- Have the source program listing available.

S IEM1637I THE THIRD ARGUMENT TO THE FUNCTION SUBSTR IN STATEMENT NUMBER xxx IS GREATER THAN THE LENGTH OF THE FIRST ARGUMENT. AN EXECUTION ERROR WILL RESULT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1638I THE THIRD ARGUMENT TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx IS GREATER THAN THE LENGTH OF THE STRING zzzz. AN EXECUTION ERROR WILL RESULT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1639I COMPILER ERROR. INCORRECT INPUT TO SUBROUTINE 6 IN MODULE IEMMF IN STATEMENT NUMBER xxx.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1640I THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz DOES NOT MATCH ANY OF THE FAMILY MEMBERS.

System Action: Terminates compilation

Programmer Response: Probable user error. Provide correct parameter description. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

W IEM1641I THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz DESCRIBES THE ENTRY NAME'S RESULT TYPE RATHER THAN ARGUMENT TYPE. IF POSSIBLE, GENERIC SELECTION WILL BE MADE ON THE BASIS OF THIS RESULT TYPE.

Programmer Response: Probable user error. Provide fuller parameter description. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1642I THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz IS NOT SUFFICIENT FOR THE PURPOSES OF GENERIC SELECTION.

System Action: Terminates compilation

Programmer Response: Probable user error. Provide fuller parameter description. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job

stream and source program listing available.

T IEM1643I COMPILER ERROR. THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz IS INCORRECTLY FORMED IN THE DICTIONARY.

System Action: Terminates compilation

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1644I COMPILER ERROR. THE GENERIC FAMILIES ASSOCIATED WITH ENTRY NAME zzzz HAVE BEEN INCORRECTLY FORMED IN THE DICTIONARY.

Explanation: The dictionary entry for one or more of the generic families is not a recognizable entry type.

System Action: Terminates compilation

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1645I THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz RESULTS IN MORE THAN ONE POSSIBLE FAMILY MEMBER SELECTION.

System Action: Terminates compilation

Programmer Response: Probable user error. Provide fuller parameter description. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1648I COMPILER ERROR. FUNCTION REFERENCE MISSING FROM TEXT IN STATEMENT NUMBER xxx

Explanation: Incorrect handling of text by previous phase.

System Action: Terminates compilation

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1649I COMPILER ERROR. INCORRECT FORMATION OF ARGUMENT LIST ASSOCIATED WITH ENTRY NAME zzzz IN STATEMENT NUMBER xxx

Explanation: Incorrect handling of text by previous phase

System Action: Terminates compilation

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM1650I COMPILER ERROR. INCORRECT HANDLING OF ARGUMENT LIST ASSOCIATED WITH ENTRY NAME zzzz IN STATEMENT NUMBER xxx

Explanation: Incorrect handling of text by previous phase

System Action: Terminates compilation

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1651I COMPILER ERROR. ARGUMENT REFERENCE MISSING FROM ARGUMENT LIST ASSOCIATED WITH ENTRY NAME zzzz IN STATEMENT NUMBER xxx

Explanation: Incorrect handling of text by previous phase

System Action: Terminates compilation

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1652I IMPLEMENTATION RESTRICTION. INVOCATIONS ARE NESTED BEYOND THE MAXIMUM PERMITTED LEVEL IN STATEMENT NUMBER xxx

Explanation: Nesting level exceeds implementation limit

System Action: Terminals compilation

Programmer Response: Probable user error. Reduce nesting level. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1654I THE GENERIC PROCEDURE zzzz IS BEING INVOKED WITHOUT AN ARGUMENT LIST IN STATEMENT NUMBER xxx

System Action: Terminates compilation

Programmer Response: Probable user error. Supply argument list. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1655I IMPLEMENTATION RESTRICTION. TOO MUCH WORKSPACE REQUIRED FOR TEMPORARY RESULTS IN STATEMENT NUMBER xxx

System Action: Terminates compilation

Programmer Response: Probable user error. Subdivide the statement in question into two or more separate statements. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the

comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM1656I COMPILER ERROR. INCORRECT INPUT TO PHASE MF FOR COMPLETION BUILT-IN FUNCTION IN STATEMENT NUMBER xxx.

Explanation: The compiler has encountered incorrect input to phase MF.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM1657I THE FILE zzzz, WHICH HAS BEEN DECLARED WITH THE COBOL OPTION, IS BEING PASSED AS AN ARGUMENT IN STATEMENT NUMBER xxx.

Explanation: (F) compiler restriction: files with the COBOL option may not be passed as arguments.

System Action: Comment and continue

Programmer Response: Probable user error. Correct source program if necessary. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1658I IN STATEMENT NUMBER xxx, zzzz IS NOT A PERMISSIBLE ARGUMENT. AN EXECUTION ERROR WILL RESULT IF THE CORRESPONDING PARAMETER IS REFERENCED

Explanation: A condition name appears as an argument in a CALL statement or function reference. This is illegal.

System Action: Attempts to pass the argument.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1670I STATEMENT NUMBER xxx HAS CAUSED A TABLE INTERNAL TO THE COMPILER TO OVERFLOW.

Explanation: Either the nesting of procedure arguments requiring dummies is too deep, or too many temporary results are required between the assignment of an argument expression to a dummy and the procedure call.

System Action: Compilation is terminated

Programmer Response: Probable user error. Reduce complexity of argument expressions. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1671I COMPILER ERROR NUMBER MP nnnn IN STATEMENT NUMBER xxx

Explanation: This is a compiler error

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM1680I COMPILER ERROR. TRIPLE OPERATOR NOT RECOGNIZED IN STATEMENT NUMBER xxx

Explanation: Illegal input from a previous phase

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1687I COMPILER ERROR. OPTIMIZED SUBSCRIPT INCORRECTLY FORMED IN STATEMENT NUMBER xxx

Explanation: Illegal input from a previous phase

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1688I COMPILER ERROR. ARRAY NAME zzzz INCORRECTLY DESCRIBED AS DEFINED IN STATEMENT NUMBER xxx

Explanation: Array incorrectly described by a previous phase as having the DEFINED attribute

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1689I COMPILER ERROR. ARRAY zzzz IS INCORRECTLY SUBSCRIPTED IN STATEMENT NUMBER xxx

Explanation: Illegal input from a previous phase

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM1692I IMPLEMENTATION RESTRICTION. SUBSCRIPT NESTED TO DEPTH GREATER THAN 50 LEVELS IN STATEMENT NUMBER xxx

Explanation: Subscript nesting exceeds fifty levels

System Action: Compilation is terminated

Programmer Response: Probable user error. Reduce amount of nesting and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job

stream and source program listing available.

T IEM1693I NUMBER OF SUBSCRIPTS ASSOCIATED WITH ARRAY zzzz IN STATEMENT NUMBER xxx IS INCORRECT.

Explanation: The number of subscripts given does not agree with the declared dimensionality of the array.

System Action: Compilation is terminated

Programmer Response: Probable user error. Add or delete subscripts as appropriate. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

W IEM1695I TRANSLATE FUNCTION IN STATEMENT NUMBER xxx HAS A CHARACTER OR BIT DUPLICATED IN ITS THIRD ARGUMENT.

Explanation: This may be a source program error.

Programmer Response: Probable user error. Check that the character or bit was intentionally duplicated. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1696I VERIFY FUNCTION IN STATEMENT NUMBER xxx HAS A CHARACTER OR BIT DUPLICATED IN ITS SECOND ARGUMENT.

Explanation: This may be a source program error.

Programmer Response: Probable user error. Check that the character or bit was intentionally duplicated. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1750I zzzz IS AN ILLEGAL OPERAND IN AN IF STATEMENT OR WHILE CLAUSE IN STATEMENT NUMBER xxx. IT HAS BEEN REPLACED BY A ZERO BIT STRING.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1751I THE IDENTIFIER zzzz IS AN ILLEGAL ARGUMENT OF THE RETURN STATEMENT NUMBER xxx AND HAS BEEN DELETED.

Explanation: Illegal arguments include arrays and structures.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1752I THE ATTRIBUTES OF THE EXPRESSION USED IN THE RETURN STATEMENT IN STATEMENT NUMBER xxx CONFLICT WITH THE ATTRIBUTES OF SOME OR ALL OF THE ENTRY POINTS OF THE CONTAINING PROCEDURE. AN EXECUTION FAILURE MAY OCCUR AT THIS STATEMENT.

Explanation: After a call to a procedure through an entry point with POINTER, AREA or data attributes, any RETURN statement encountered must return a value of type POINTER or AREA or of a data type compatible with the data attributes of the entry point.

System Action: The ERROR condition is raised

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1753I THE EXPRESSION USED IN THE RETURN STATEMENT IN STATEMENT NUMBER xxx AND THE ATTRIBUTES OF THE CONTAINING PROCEDURE ARE INCOMPATIBLE. EXECUTION OF THIS STATEMENT WILL RESULT IN A FAILURE.

Explanation: After a call to a procedure through an entry point with POINTER, AREA or data attributes, any RETURN statement encountered must return a value of type POINTER or AREA or of a data type compatible with the data attributes of the entry point.

System Action: The ERROR condition is raised

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1754I THE EXPRESSION USED IN THE RETURN STATEMENT IN STATEMENT NUMBER xxx IS INVALID

Explanation: The only permitted arguments are data types STRING, POINTER, and AREA.

System Action: Raise ERROR condition on execution of the statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1755I OPTION SPECIFICATION CONTAINS INVALID ARGUMENT, DEFAULT USED FOR SORMGIN.

Explanation: This message is written directly on SYSPRINT. The compiler found that an argument to the SORMGIN option was either zero or greater than 100.

System Action: The default interpretation for SORMGIN, as set at system generation, is used.

Programmer Response: Probable user error. Correct the erroneous argument, and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1790 DATA CONVERSIONS WILL BE DONE BY SUBROUTINE CALL IN THE FOLLOWING STATEMENTS yyyy

Programmer Response: Check to see if the conversion can be avoided or performed in line

S IEM1793I ILLEGAL ASSIGNMENT OR CONVERSION IN STATEMENT NUMBER xxx. EXECUTION WILL RAISE THE ERROR CONDITION.

Explanation: Illegal assignment or conversion in source statement, e.g. label to arithmetic.

System Action: An instruction is compiled which will cause execution to abort if the statement is executed

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1794I COMPILER ERROR IN STATEMENT NUMBER xxx PHASE OE.

Explanation: Compiler error caused by input text in bad format

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the

comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

S IEM1795I INVALID ITEM IN FREE STATEMENT NUMBER xxx

Explanation: Variable in FREE statement is either not CONTROLLED or not at level 1

System Action: Error condition and message given at object time

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1796I ASSIGNMENT OF AN ILLEGAL LABEL CONSTANT IN STATEMENT NUMBER xxx.

Explanation: The label constant does not appear in the value list in the DECLARE statement for the label variable.

System Action: Accepts label constant as if in value list and continues compilation.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1797I CONVERSION OF NULL VALUES IN POINTER/OFFSET ASSIGNMENTS IS INVALID. NULLO HAS BEEN REPLACED BY NULL, OR NULL BY NULLO, IN STATEMENT NUMBER xxx

Explanation: A NULLO offset type constant has been assigned to a pointer, or a NULL pointer type constant to an offset. Conversion of null values is not allowed. The constant type has been corrected.

System Action: The assignment is unaffected

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1800I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO FLOATING-POINT. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

System Action: Truncates result.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1801I AN ERROR HAS OCCURRED IN THE CONVERSION TO FLOATING-POINT OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1802I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO FIXED BINARY. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1803I AN ERROR HAS OCCURRED IN THE CONVERSION TO FIXED BINARY OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1804I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO FIXED DECIMAL. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

System Action: Truncates result.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1805I AN ERROR HAS OCCURRED IN THE CONVERSION TO FIXED DECIMAL OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USE OF THIS CONSTANT.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1806I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO DECIMAL NUMERIC FIELD. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1807I AN ERROR HAS OCCURRED IN THE CONVERSION TO DECIMAL NUMERIC FIELD OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1808I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO STERLING NUMERIC FIELD. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1809I AN ERROR HAS OCCURRED IN THE CONVERSION TO STERLING NUMERIC FIELD OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

Programmer Response: Probable user error. Change the constant and check its use in

the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1810I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO BIT STRING. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1811I AN ERROR HAS OCCURRED IN THE CONVERSION TO BIT STRING OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1812I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO CHARACTER STRING. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1813I AN ERROR HAS OCCURRED IN THE CONVERSION TO CHARACTER STRING OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1814I AN ERROR HAS OCCURRED IN THE CONVERSION OF THE CONSTANT yyyy TO PICTURED CHARACTER STRING. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1815I AN ERROR HAS OCCURRED IN THE CONVERSION TO PICTURED CHARACTER STRING OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

Programmer Response: Probable user error. Change the constant and check its use in the given statement and elsewhere. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1816I zzzz USED IN FILE OPTION IN STATEMENT NUMBER xxx IS NOT A FILE. OPTION HAS BEEN IGNORED. EXECUTION ERROR WILL RESULT

Explanation: Dictionary reference of file triple was

not file constant or file parameter code.

System Action: Ignores option, but continues to scan statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1817I INVALID KEYTO OPTION zzzz IGNORED IN STATEMENT NUMBER xxx

Explanation: KEYTO option must be scalar character string variable.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1818I zzzz USED IN KEY/KEYFROM OPTION IN STATEMENT NUMBER xxx IS NOT A SCALAR. OPTION IGNORED.

System Action: Ignores option but continues scan of statement

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1819I zzzz USED IN THE IGNORE OPTION IN STATEMENT NUMBER xxx IS NOT A SCALAR. OPTION IGNORED.

System Action: Ignores option but continues scan of statement

Programmer Response: Probable user error. Correct IGNORE variable. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1823I COMPILER ERROR DETECTED IN PHASE NJ/NK.

Explanation: NJ/NK found some unexpected input. Register 9 in dump will indicate cause of error.

System Action: Terminates compilation

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM1824I OPTIONS IN OPEN STATEMENT NUMBER xxx ARE IN CONFLICT WITH PAGESIZE AND/OR LINESIZE.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1825I INVALID REPLY OPTION IGNORED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1826I INVALID MESSAGE IN DISPLAY STATEMENT NUMBER xxx. STATEMENT IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1827I INVALID ARGUMENT TO DELAY STATEMENT NUMBER xxx. STATEMENT IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1828I COMPILER ERROR. INCORRECT NUMBER OF TMPDS FOLLOWING ZERO OPERAND IN STATEMENT NUMBER xxx

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM1829I INVALID SCALAR EXPRESSION OPTION IN WAIT STATEMENT NUMBER xxx. MAXIMUM EVENT COUNT GIVEN.

Explanation: The optional scalar expression in the WAIT statement cannot be converted to an integer.

System Action: The number of event names in the list is assumed as the event count.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1830I COMPILER ERROR. INCORRECT INPUT TO PHASE NG IN WAIT STATEMENT NUMBER xxx.

Explanation: The compiler has encountered incorrect input to phase NG and cannot continue.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM1831I INVALID KEYTO OPTION IN STATEMENT NUMBER xxx

Explanation: KEYTO option must be scalar character-string variable

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1832I INVALID PAGE OPTION IGNORED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1833I INVALID LINE OPTION IGNORED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1834I MULTIPLE COPY OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE FIRST ONE IS USED.

System Action: The first option only is used

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1835I INVALID FILE OPTION IGNORED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1836I INVALID STRING OPTION IGNORED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1837I NO FILE OR STRING SPECIFIED IN STATEMENT NUMBER xxx. STATEMENT IGNORED.

Explanation: No FILE or STRING given in GET/PUT statement

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1838I INVALID TITLE OPTION IGNORED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1839I INVALID IDENT OPTION IGNORED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1840I INVALID LINESIZE OPTION IGNORED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1841I INVALID PAGESIZE OPTION IGNORED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1843I NO FILE SPECIFIED IN OPEN/CLOSE STATEMENT NUMBER xxx. ANY OPTIONS ARE IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM1844I COMPILER ERROR. INCORRECT NUMBER OF TMPDS FOLLOWING ZERO OPERAND IN STATEMENT NUMBER xxx

System Action: Compilation is aborted

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

E IEM1845I MULTIPLE DATA SPECIFICATIONS IGNORED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1846I INVALID SKIP OPTION IGNORED IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1847I NO DATA SPECIFICATIONS GIVEN FOR GET STATEMENT NUMBER xxx. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1848I NO DATA SPECIFICATIONS OR PRINT OPTIONS GIVEN FOR PUT STATEMENT NUMBER xxx. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1849I THE USE OF THE BUILT-IN FUNCTION NULL IN STATEMENT NUMBER xxx IS INVALID; NULLO HAS BEEN SUBSTITUTED. CHECK ALL SIMILAR USES OF NULL.

System Action: Substitute NULLO

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1850I THE USE OF THE BUILT-IN FUNCTION NULLO IN STATEMENT NUMBER xxx IS INVALID; NULL HAS BEEN SUBSTITUTED. CHECK ALL SIMILAR USES OF NULLO.

System Action: Substitute NULL

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1860I THE ILLEGAL ITEM zzzz HAS BEEN DELETED FROM THE I/O DATA LIST IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1861I AN ILLEGAL TEMPORARY RESULT OR SUBSCRIPTED ELEMENT HAS BEEN DELETED FROM THE I/O DATA LIST IN STATEMENT NUMBER xxx

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1862I AN EXPRESSION OR FUNCTION INVOCATION IS AN ILLEGAL DATA ITEM AND HAS BEEN DELETED FROM THE DATA-DIRECTED I/O STATEMENT NUMBER xxx.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM1870I THE FORMAT LIST IN STATEMENT NUMBER xxx CONTAINS NO DATA FORMAT ITEMS AND WILL BE EXECUTED ONCE IF THE STATEMENT IS INVOKED.

System Action: At execution time, on finding no data format items, control passes out of the statement at the end of the format list.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1871I IN STATEMENT NUMBER xxx THE FORMAT LIST CONTAINS AN E OR F FORMAT ITEM WITH AN ILLEGAL SPECIFICATION. THE FORMAT ITEM HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1872I IN STATEMENT NUMBER xxx AN E FORMAT ITEM HAS A FIELD WIDTH WHICH WOULD NOT PERMIT PRINTING OF A MINUS SIGN.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1873I IMPLEMENTATION RESTRICTION. IN STATEMENT NUMBER xxx AN A, B OR CONTROL FORMAT ITEM SPECIFIES AN EXCESSIVE LENGTH WHICH HAS BEEN REPLACED BY THE MAXIMUM OF 32,767.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM1874I IN STATEMENT NUMBER xxx AN INPUT STATEMENT CONTAINS A FORMAT ITEM WHICH MAY BE USED ONLY IN OUTPUT STATEMENTS.

Explanation: PAGE, SKIP, LINE, COLUMN, and format items A and B with no width specification, may be used only for output.

System Action: Invalid format item deleted

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM1875I IN STATEMENT NUMBER xxx AN E FORMAT ITEM HAS AN ILLEGAL SPECIFICATION IF USED FOR AN OUTPUT DATA ITEM.

Explanation: The specification violates the restriction that the field width w must be greater than s+n+2.

System Action: There will be an error at execution time.

Programmer Response: Probable user error. Correct specification. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM2304I COMPILER ERROR. DICTIONARY ENTRY zzzz UNRECOGNIZED IN STATIC CHAIN.

Explanation: Due to a compiler error, a dictionary entry with an unrecognized code byte has been found in the static chain.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM2305I COMPILER ERROR. DOPE VECTOR REQUESTED BY NON-STRING, NON-STRUCTURE MEMBER zzzz

Explanation: Due to a compiler error, the allocation of a dope vector has been requested for an item which should never require one.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM2352I THE AUTOMATIC VARIABLES IN THE BLOCK HEADED BY STATEMENT NUMBER xxx ARE MUTUALLY DEPENDENT. STORAGE CANNOT BE ALLOCATED.

Explanation: This message is generated when a number of automatic variables are mutually dependent. It is not then possible to allocate storage in order of dependency.

System Action: Compilation is terminated

Programmer Response: Probable user error. Rewrite statement, eliminating mutual dependency. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM2650I IMPLEMENTATION RESTRICTION. OPTIMIZATION TABLE OVERFLOWED WHILE PROCESSING STATEMENT NUMBER xxx.

System Action: Compilation terminated.

Programmer Response: Probable user error. Re-compile with one of the following changes:

1. Use a larger partition or region

2. Specify OPT=0 or 1

3. Reduce the number of subscripts in the DO loop that contains the statement indicated.

If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

T IEM2660I COMPILER ERROR IN INPUT TO PHASE RD. IN STATEMENT NUMBER xxx A PREVIOUS PHASE HAS GENERATED A LABEL NUMBER GREATER THAN THE MAXIMUM SHOWN.

System Action: Compilation is terminated.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job

stream and source program listing available.

T IEM2661I COMPILER ERROR. INTERNAL TABLE ENTRY IN PHASE RD IS INCORRECT.

Explanation: An entry in the internal table of compiler-generated labels does not point to a label in text.

System Action: Compilation is terminated.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.
- Recompilation should also be tried without the OPT=2 option.

T IEM2700I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. SPECIAL ASSIGNED REGISTER IN FORMAT/DATA LIST CODE CANNOT BE FOUND.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM2701I COMPILER ERROR. PHASE IEMRF, STATEMENT NUMBER xxx. PSTOR GREATER THAN 32K.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM2702I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. BCT WITHOUT DICTIONARY REFERENCE AS DESTINATION.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM2703I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. LINK REGISTER IN BALR IS NOT ASSIGNED.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM2704I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. 'USNG' ITEM DOES NOT HAVE ASSIGNED REGISTER.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM2705I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. DROPPED REGISTER NOT ACTIVE.

Explanation: Register number in field in DROP item is not in register table nor in storage.

System Action: Continues compilation, ignoring DROP. Execution is inhibited.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,RF)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM2706I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. NOT ALL REGISTERS IN 'DRPL' ITEM CAN BE FOUND.

System Action: Ignores DRPL item and continues

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,RF)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM2707I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. NOT ALL SYMBOLIC REGISTERS DROPPED AT END OF PROCEDURE OR BEGIN BLOCK.

Explanation: One or more symbolic registers have been used in the PROCEDURE or BEGIN block, but no corresponding DROP has occurred.

System Action: Inserts in listing at end of block: the register number, the offset from register 9 at which the register is stored, and the words 'ERROR STOP'.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,RF)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM2708I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. ASSIGNED REGISTER USED IN SOURCE FIELD IS NOT INITIALIZED.

Explanation: The assigned register should have a previous value (e.g. X in AR X,Y or L Y,10(X) etc.), but none can be found.

System Action: Register 13 is used instead of the correct number, and compilation is continued.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,RF)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

S IEM2709I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. SYMBOLIC REGISTER SHOULD HAVE PREVIOUS VALUE, BUT HAS NOT.

Explanation: Register X in an instruction such as AR X,Y, or L Y,10(X), has not been set up previously.

System Action: Inserts register 12 and continues compilation.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,RF)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM2710I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. MORE THAN ONE REGISTER PAIR REQUIRED IN AN INSTRUCTION.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM2711I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. ASSIGNED REGISTER IS STILL IN USE AT THE START OF A PROCEDURE.

Explanation: Assigned register status should be zero at the start of each procedure.

System Action: Drops the assigned register and continues compilation

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,RF)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM2712I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. IPRM/IPRM' OR EPRM/EPRM' PAIRS ARE NOT MATCHED IN PREVIOUS STATEMENT.

System Action: Compilation is continued

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,RF)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM2816I ILLEGAL ENVIRONMENT OPTION IN STATEMENT NUMBER xxx

System Action: Remainder of environment attributes ignored.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM2817I COMPILER ERROR. INVALID ATTRIBUTE CODE IN STATEMENT NUMBER xxx

Explanation: An invalid attribute marker has been found in the dictionary entry corresponding to the file attributes in the statement specified

System Action: Ignores the rest of the entry

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,GA)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM2818I CONFLICTING ATTRIBUTE IN STATEMENT NUMBER xxx IGNORED.

Explanation: An attribute other than 'ENVIRONMENT' clashes with previously declared attributes in the specified statement

System Action: Ignores this attribute.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM2819I ERRONEOUS USE OF PARENTHESIS IN ENVIRONMENT OPTION IN STATEMENT NUMBER xxx

Explanation: Misplaced parenthesis in ENVIRONMENT attribute

System Action: Remainder of ENVIRONMENT attribute ignored

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM2820I ERRONEOUS USE OF COMMA IN ENVIRONMENT OPTION IN STATEMENT NUMBER xxx

Explanation: Misplaced comma in ENVIRONMENT attribute

System Action: Remainder of ENVIRONMENT attribute ignored

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM2821I ILLEGAL CHARACTER IN KEYWORD IN ENVIRONMENT OPTION IN STATEMENT NUMBER xxx

Explanation: Invalid keyword in ENVIRONMENT attribute

System Action: Remainder of ENVIRONMENT attribute ignored

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM2822I FIELD TOO LARGE IN ENVIRONMENT OPTION IN STATEMENT NUMBER xxx

Explanation: Field in item in ENVIRONMENT attribute too large

System Action: Remainder of ENVIRONMENT attribute ignored

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM2823I ERROR IN FORMAT OF ENVIRONMENT ATTRIBUTE IN STATEMENT NUMBER xxx

Explanation: Format of item in ENVIRONMENT attribute incorrect

System Action: Remainder of ENVIRONMENT attribute ignored

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM2824I CONFLICT BETWEEN ENVIRONMENT ATTRIBUTE AND OTHER ATTRIBUTES IN STATEMENT NUMBER xxx

Explanation: An option in the ENVIRONMENT attribute clashes with either another ENVIRONMENT option or with a declared attribute.

System Action: Remainder of ENVIRONMENT attribute ignored

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM2825I CONFLICTING OPTIONS IN ENVIRONMENT ATTRIBUTE IN STATEMENT NUMBER xxx. REST OF ENVIRONMENT IGNORED.

System Action: DECLARE control block is constructed from attributes which have already been processed. The rest are ignored.

Programmer Response: Probable user error. Correct ENVIRONMENT option. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM2826I IMPLEMENTATION RESTRICTION. DIRECT FILE zzzz DECLARED IN STATEMENT NUMBER xxx MUST HAVE AN ORGANIZATION SUBFIELD IN THE ENVIRONMENT ATTRIBUTE.

System Action: No compile-time action, but execution will fail.

Programmer Response: Probable user error. Provide ENVIRONMENT attribute. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM2827I A D COMPILER OPTION HAS BEEN DECLARED IN THE ENVIRONMENT LIST IN STATEMENT NUMBER xxx. IT HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM2828I ENVIRONMENT OPTIONS CLTASA AND CLT360 HAVE BOTH BEEN DECLARED IN STATEMENT NUMBER xxx. THE SECOND ONE LISTED WILL BE IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM2829I IN STATEMENT NUMBER xxx THE PARAMETER SPECIFIED IN THE INDEXAREA OPTION IS GREATER THAN 32767 AND HAS BEEN IGNORED.

Explanation: If the parameter is not specified or is outside the permitted range, data management uses as much main storage as is required for the master index.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM2830I FILE DECLARED TRANSIENT IN STATEMENT NUMBER xxx DOES NOT HAVE MANDATORY ENVIRONMENT OPTION G OR R.

System Action: Compilation terminated.

Programmer Response: Probable user error. Correct file declaration. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM2831I THE NCP VALUE IN STATEMENT NUMBER xxx EITHER IS NOT AN INTEGER CONSTANT OR LIES OUTSIDE THE PERMITTED RANGE OF 1 TO 99. A VALUE OF 1 HAS BEEN ASSUMED.

Programmer Response: Probable user error. Correct the NCP value and re-compile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM2833I COMPILER ERROR. OPERAND OF CL OR SL NOT LABEL.

Explanation: The dictionary entry referenced after a compiler label or statement label marker in the text is not in fact a label

System Action: The label definition is ignored.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,TF)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM2834I COMPILER ERROR. INVALID PSEUDO-CODE OPERATION.

Explanation: The input text contains a marker which is not valid

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM2835I COMPILER ERROR. SUBSCRIPTED LABEL CHAIN ERROR.

Explanation: Subscripted labels in the source program result in the creation of chains of dictionary entries. An error in the chaining causes this message to appear.

System Action: The label definition is ignored

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,TF)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM2836I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE.

Explanation: Not enough scratch core is available for the generated label number table created by this phase. The condition arises when a large number of such labels have been used, and this in turn is related to the size of the program.

System Action: The compilation is terminated

Programmer Response: Probable

user error. Break down the program into smaller modules. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM2837I COMPILER ERROR. MULTIPLY DEFINED LABEL OR INVALID LABEL NUMBER.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM2838I THE TELEPROCESSING FORMAT OPTION IN STATEMENT NUMBER xxx CONTAINS NO CONSTANT. A VALUE OF ZERO HAS BEEN ASSUMED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM2840I NUMERIC FIELD IN ENVIRONMENT OPTION FOR FILE zzzz DECLARED IN STATEMENT NUMBER xxx IS NOT IN PARENTHESES AND HAS BEEN DELETED.

Explanation: If an environment option requires a numeric field, then that field must be enclosed in parentheses.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM2865I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM CONTAINS TOO MANY BLOCKS AND/OR CONTROLLED VARIABLES.

Explanation: The compiler allocates a pseudo-register entry for each block and CONTROLLED variable in the source program. The maximum number of such entries is 1,024

System Action: No pseudo-registers are allocated for items after the limit has been reached.

Programmer Response: Probable user error. Reduce number of blocks, or CONTROLLED variables, in program to less than 1,025. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM2866I THIS PL/I COMPILATION HAS GENERATED EXTERNAL NAMES IN WHICH THE FIRST LEADING CHARACTER OF THE EXTERNAL PROCEDURE NAME HAS BEEN REPLACED BY A SPECIAL CHARACTER.

Explanation: The external procedure name with its first character changed is being used as a base for generating names for External Symbol Dictionary entries. If the same thing happens in another compilation, and the two are then joined by the Linkage Editor, two External Symbol Dictionary entries may have the same name.

System Action: None

E IEM2867I IMPLEMENTATION RESTRICTION. EXTERNAL NAME zzzz HAS BEEN TRUNCATED TO 7 CHARACTERS.

Explanation: External identifiers are restricted to 7 characters

System Action: Name of ESD entry truncated by taking first 4 and last 3 characters; phase then carries on normally.

Programmer Response: Probable user error. Shorten the name. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM2868I THIS PL/I COMPILATION HAS GENERATED EXTERNAL NAMES IN WHICH THE SECOND LEADING CHARACTER OF THE EXTERNAL PROCEDURE NAME HAS BEEN REPLACED BY A SPECIAL CHARACTER.

Explanation: The external procedure with its second character changed is being used as a base for generating names for External Symbol Dictionary entries. If the same thing happens in another compilation, and the two are then joined by the Linkage Editor, two External Symbol Dictionary entries may have the same name.

System Action: None

T IEM2881I COMPILER ERROR IN STATEMENT NUMBER xxx. INVALID PSEUDO-CODE OPERATION.

Explanation: The input text contains a marker which is not valid.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,TT)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM2882I COMPILER ERROR IN STATEMENT NUMBER xxx. OPERAND OF DC CODE INVALID.

Explanation: The operand of a DCA4 pseudo-code item is not valid - the operand should always be relocatable.

System Action: An offset of zero is assembled into the text.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,TT)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM2883I COMPILER ERROR IN STATEMENT NUMBER xxx. INVALID REQUEST FOR RELOCATABLE TEXT.

Explanation: The operand of a branch instruction has been found to require relocation.

System Action: An offset of zero is assembled into the text.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,TT)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM2888I COMPILER ERROR IN STATEMENT NUMBER xxx. UNDEFINED LABEL.

Explanation: No offset has been assigned to a label generated by the compiler.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options

'S,DP=(PIE,TT)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

S IEM2897I IMPLEMENTATION RESTRICTION. QUALIFIED NAME zzzz LONGER THAN 256 CHARACTERS.

Explanation: The fully qualified name of the variable indicated will not fit into its Symbol Table entry

System Action: Leaves Symbol Table entry incomplete and carries on with the initialization of the Static Internal control section.

Programmer Response: Probable user error. Shorten the qualified name. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM2898I DATA-DIRECTED 'GET/PUT' STATEMENT WITH NO LIST IN A PROCEDURE OR BEGIN BLOCK WHICH HAS NO DATA VARIABLES.

System Action: Zeros are inserted in the argument list for the call to the library routine to 'GET/PUT DATA', and compilation continues.

Programmer Response: Probable user error. Correct GET/PUT statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM2899I INITIALIZATION SPECIFIED FOR TOO FEW ELEMENTS IN STATIC ARRAY zzzz

System Action: Initialization terminated when end of initial string is found.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM2900I INITIALIZATION SPECIFIED FOR TOO MANY ELEMENTS IN STATIC ARRAY zzzz

System Action: Initialization is terminated when every element has been initialized.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM2913I COMPILER ERROR. INVALID PSEUDO-CODE OPERATION.

Explanation: The input text contains a marker which is not valid

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM3088I -3213I THE CONFLICTING ATTRIBUTE aaaa HAS BEEN IGNORED IN THE DECLARATION OF IDENTIFIER yyyy IN STATEMENT NUMBER xxx

Explanation: The attribute given in the message conflicts with another attribute declared for the same identifier, or is invalid for that identifier.

System Action: The attribute given in the message is ignored.

Programmer Response: Probable user error. Correct program and recompile. If the problem

recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM3584I AN UNBALANCED NUMBER OF PARENTHESES HAS BEEN DETECTED WITHIN A STATEMENT AT OR NEAR STATEMENT NUMBER xxx

Explanation: An occurrence of a comma immediately followed by a period at or near the given statement has been taken as a statement delimiter. The statement contains an unbalanced number of parentheses.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM3839I COMPILER ERROR. INVALID ERROR MESSAGE CHAINS.

Explanation: Compiler is unable to print error diagnostics.

System Action: Compilation is terminated.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3841I I/O ERROR ON SEARCHING DIRECTORY.

Explanation: This message is written directly on SYSPRINT. A permanent I/O error was detected when an attempt was made to search the directory of the library containing the compiler.

System Action: Compilation is terminated

Programmer Response: Check the directory and re-attempt compilation. If the input/output error persists, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement was included for the failing job step.
- Have the associated job stream and source program listing available.

T IEM3842I COMPILER ERROR AT STATEMENT NUMBER xxx. ALL TEXT BLOCKS IN CORE ARE BUSY. REFERENCED BLOCK CANNOT BE BROUGHT INTO CORE.

Explanation: All blocks in core have become busy. Compiler cannot continue since an external block cannot be read in.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3843I COMPILER ERROR AT STATEMENT NUMBER xxx. ATTEMPTED USE OF PHASE yyy OF ZDABRF WITH BLOCK NOT IN CORE.

Explanation: Referenced block is not in core

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3844I IMPLEMENTATION RESTRICTION. DICTIONARY ENTRY FOR STRING CONSTANT, PICTURE, DOPE VECTOR, OR STATIC INITIAL STRING IS TOO LONG FOR THIS SIZE OPTION.

Explanation: The dictionary entry cannot be made because it is larger than the dictionary block size of this compilation.

System Action: Terminates compilation

Programmer Response: Probable user error. If the entry is smaller than 16K bytes, increase the size option to give dictionary blocks bigger than the required entry. The largest dictionary block size is 16K, and in cases where a larger entry was required, it is necessary to avoid the source program feature which caused the excessive dictionary entry request. In general, write the source program in such a way that the required function is performed at execution time rather than compile time. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3845I COMPILER ERROR AT STATEMENT NUMBER xxx. TEXT BLOCK REFERENCED BY PHASE yyy IS NOT IN CORE.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3846I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE BY STATEMENT NUMBER xxx. ALL TEXT BLOCKS FULL WHEN PHASE yyy IS BEING EXECUTED.

Explanation: There is no more space for text in this environment

System Action: Compilation is terminated

Programmer Response: Probable user error.

1. Subdivide the program and recompile.
2. Increase the SIZE option for the compiler to obtain larger text blocks, and recompile.
3. If OPT=2 has been specified for this compilation, recompile specifying OPT=1 or OPT=0.

All three possibilities may be tried together. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job

stream and source program listing available.

T IEM3847I COMPILER ERROR AT STATEMENT NUMBER xxx. PHASE yyy HAS REQUESTED MORE THAN 4K OF SCRATCH CORE.

Explanation: Request for scratch core exceeds 4096 bytes

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3848I COMPILER ERROR AT STATEMENT NUMBER xxx. PHASE yyy HAS REQUESTED A RELEASE OF UNALLOCATED CORE.

Explanation: Attempt to release unallocated scratch core

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3849I COMPILER ERROR. PHASE yy IN RELEASE LIST IS NOT IN PHASE DIRECTORY.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3850I COMPILER ERROR. PHASE yy IN LOAD LIST IS NOT IN PHASE DIRECTORY.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3851I COMPILER ERROR. PHASE yy NOT MARKED. IT IS LOADED.

Explanation: An unmarked phase is loaded

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3852I COMPILER ERROR AT STATEMENT NUMBER xxx. BLOCK REFERENCED BY PHASE yyy IS NOT IN USE. COMPILER CANNOT CONTINUE.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3853I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE. DICTIONARY IS FULL.

Explanation: This message is written directly on SYSPRINT.

System Action: Compilation is terminated

Programmer Response: Probable user error. Subdivide into more than one program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3855I ERROR IN PHASE yy.

Explanation: This message is written directly on SYSPRINT. A compiler error has been discovered during the printing of compile-time diagnostic messages (if phase quoted in message is BM) or of source-program diagnostic messages (if phase is XA).

System Action: Compilation is terminated. Note that only the diagnostic message output is incomplete. All other output files have been generated satisfactorily.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3856I COMPILER ERROR. PROGRAM CHECK TYPE nnnn HAS OCCURRED IN PHASE yy AT OR NEAR STATEMENT NUMBER xxx

Explanation: A program check has occurred during compilation. This is due to a compiler failure which may have been exposed by an error in the source code. The check number is the code for a program interrupt as follows:

Code	Cause of Program Interrupt
1	Operation
2	Privileged operation
3	Execute
4	Protection
5	Addressing
6	Specification
7	Data

System Action: Compilation is terminated

Programmer Response: Check source code carefully. If an error is found, correcting it may enable compilation to be completed successfully. Whether or not an error is found do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3857I COMPILER ERROR. ATTEMPT TO PASS CONTROL TO AN UNNAMED PHASE. AN UNMARKED PHASE HAS BEEN ENCOUNTERED.

Explanation: An unmarked phase

has been encountered. Compiler cannot continue

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3858I COMPILER ERROR. REQUESTED OR UNWANTED PHASE NOT IN PHASE DIRECTORY.

Explanation: Request to mark a phase which is not in phase directory. Compiler cannot continue.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3859I INSUFFICIENT CORE IS AVAILABLE TO CONTINUE THIS COMPILATION.

Explanation: An attempt is being made to expand the number of text blocks in core. The GETMAIN routine has failed to get the core. This will only occur where less than 45,056 bytes are available to the compiler, or when the SIZE option has been given too large a value. This message is written directly onto SYSPRINT.

System Action: Compilation is terminated. Message IEM3865I may follow.

Programmer Response: Probable user error. Check the SIZE option and check that the required core is available in the system on which the compilation is being run. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

E IEM3860I I/O ERROR ON SYSIN. RECORD ACCEPTED AS INPUT.

Explanation: The error may be a machine error, or, if SYSIN is a card reader, there may be a hole pattern which does not represent a valid System/360 character (validity check).

System Action: The error message number is printed in the source listing before the record in error. The record is accepted as input.

Programmer Response: If SYSIN is a card reader, check that every column of the indicated card contains a valid code. If the input/output error persists, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that

MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement was included for the failing job step.

- Have the associated job stream and source program listing available.

T IEM3861I I/O ERROR ON SYSLIN. GENERATION OF LOAD FILE IS TERMINATED.

Programmer Response: Check DD card and recompile. If the input/output error persists, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement was included for the failing job step.
- Have the associated job stream and source program listing available.

T IEM3862I I/O ERR,jobname,stepname,unit address,device type,SYSPRINT, operation attempted,error description
IEM3862I CONT- ,**********,access method (see Note 1) relative block number (decimal),access method (see Note 2) actual track address and block number (BBCCHHR in hexadecimal format), access method (see Note 3)

Note: IEM3862I should always be preceded by message IEA000I.

Explanation: This is an online message written to the operator. There is an I/O error on SYSPRINT. The compiler cannot continue. The information in the message is provided by the operating system. The device type field contains either:

UR - unit record device, or

TA - magnetic-tape device, or

DA - direct-access device.

Notes: 1. For a unit record device

2. For a magnetic-tape device

3. For a direct-access device

System Action: Compilation is terminated. (No further printing takes place.)

Programmer Response: Check DD statement and recompile. If the input/output error persists, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement was included for the failing job step.
- Have the associated job stream and source program listing available.

For users with MVT/MFT and with SYSPRINT assigned to a SYSOUT queue on a direct-access device, have the unit or pack containing the data set checked.

T IEM3863I I/O ERROR ON SYSPUNCH. GENERATION OF OBJECT DECK IS TERMINATED.

Programmer Response: Check DD card and recompile. If the input/output error persists, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the

comments which precede all the IEMnnnnI messages.)

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement was included for the failing job step.
- Have the associated job stream and source program listing available.

T IEM3864I I/O ERROR ON SYSUT1

Explanation: This message is written directly on SYSPRINT. There is an I/O error on SYSUT1. The compiler cannot continue.

System Action: Compilation is terminated

Programmer Response: Check DD card and recompile. If the input/output error persists, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement was included for the failing job step.
- Have the associated job stream and source program listing available.

T IEM3865I ERROR IN COMPILER ABORT

Explanation: This message is written directly on SYSPRINT. The compiler has tried twice to abort and cannot do so. Compilation will therefore terminate without the production of any further diagnostic messages.

System Action: Compilation is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM3872I I/O ERROR ON SYSUT3

Explanation: This message is written directly on SYSPRINT. There is an I/O error on SYSUT3. The compiler cannot continue.

System Action: Compilation is terminated

Programmer Response Check DD card and recompile. If the input/output error persists, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement was included for the failing job step.
- Have the associated job stream and source program listing available.

E IEM3873I I/O ERROR ON SYSUT3. RECORD ACCEPTED AS INPUT.

System Action: The error message number is printed in the source listing before the record in error. The record is accepted as input.

Programmer Response: Check DD card and recompile. If the input/output error persists, do the following before calling IBM for programming support:

- Recompile the program with

compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement was included for the failing job step.
- Have the associated job stream and source program listing available.

T IEM3874I UNABLE TO OPEN SYSIN

Explanation: This message is written directly on SYSPRINT. Unable to open SYSIN. The compiler cannot continue.

System Action: Compilation is terminated

Programmer Response: Probable user error. Check SYSIN DD card and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM3875I UNABLE TO OPEN SYSLIN. THE LOAD OPTION HAS BEEN DELETED

Explanation: The SYSLIN data set could not be opened.

System Action: The NOLOAD option is forced and compilation continues.

Programmer Response: Probable user error. Check DD statement for SYSLIN if LOAD is requested, else specify NOLOAD, and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM3876I UNABLE TO OPEN SYSPRINT

Explanation: This is an on-line message. It is written to operator. Unable to open SYSPRINT. The compiler cannot continue

System Action: Compilation is terminated

Programmer Response: Probable user error. Check SYSPRINT DD card and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was

included for the failing job step.

- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM3877I UNABLE TO OPEN SYSPUNCH. THE DECK/MACDCK OPTION HAS BEEN DELETED

Explanation: The SYSPUNCH data set could not be opened.

System Action: The NODECK and NOMACDCK options are forced and compilation continues.

Programmer Response: Probable user error. Check SYSPUNCH DD statement if LOAD is requested, else specify NODECK, NOMACDCK, and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM3878I UNABLE TO OPEN SYSUT1. COMPILATION CANNOT CONTINUE.

System Action: Compilation is terminated

Programmer Response: Probable user error. Check SYSUT1 DD card and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM3880I UNABLE TO OPEN SYSUT3

Explanation: This message is written directly on SYSPRINT. Unable to open SYSUT3. The compiler cannot continue.

System Action: Compilation is terminated

Programmer Response: Probable user error. Check SYSUT3 DD card and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

S IEM3888I SYSPUNCH BLOCKSIZE NOT A MULTIPLE OF 80. THE DECK AND

MACDCK OPTIONS HAVE BEEN DELETED.

Explanation: On opening SYSPUNCH, the blocksize definition, either on the DD card or in the data-set label, was not a multiple of 80.

System Action: The DECK and MACDCK options are deleted

Programmer Response: Probable user error. Correct the blocksize definition and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

S IEM3889I SYSLIN BLOCKSIZE NOT A MULTIPLE OF 80. THE LOAD OPTION HAS BEEN DELETED.

Explanation: On opening SYSLIN, the blocksize definition, either on the DD card or in the data-set label, was not a multiple of 80.

System Action: The LOAD option is deleted

Programmer Response: Probable user error. Correct the blocksize definition and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have associated job stream, source and program listing available.

W IEM3890I NO RECFM GIVEN FOR SYSIN. U TYPE RECORDS ARE ASSUMED

Explanation: No RECFM definition has been found in the DCB parameter of the SYSIN DD card or the data-set label.

System Action: Compilation proceeds assuming U type records. U format will be specified in the data-set label when SYSIN is closed.

Programmer Response: Probable user error. Check RECFM definition on SYSIN DD card and rerun if necessary. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM3891I SYSIN BLOCKSIZE IS TOO LARGE

Explanation: On opening SYSIN for unblocked records, a blocksize of greater than 100 has been specified either in the DCB parameter of the SYSIN DD card or in the data-set label.

System Action: Compilation is terminated

Programmer Response: Probable user error. Change the invalid blocksize definition and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM3893I SYSIN BLOCKSIZE NOT A MULTIPLE OF RECORD LENGTH.

Explanation: On opening SYSIN for FB format records, it has been noted that the blocksize definition, either on the DD card or in the data set label, is not an exact multiple of the record length.

System Action: Compilation is terminated.

Programmer Response: Probable user error. Correct the blocksize definition and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM3894I SYSIN BLOCKSIZE NOT EQUAL TO RECORD LENGTH.

Explanation: On opening SYSIN for F format records, the block size and record-length definitions, either on the DD card or in the data-set label, were found to be unequal.

System Action: Compilation is terminated.

Programmer Response: Probable user error. Change the incorrect definition and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM3895I SYSIN RECORD LENGTH TOO LARGE

Explanation: On opening SYSIN for F format records, a record length definition, either on the DD card or in the data-set label, was found to be greater than 100.

System Action: Compilation is terminated.

Programmer Response: Probable user error. Correct the record-length definition and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM3896I SYSPRINT BLOCKSIZE IS NOT OF FORM 4+N*125

Explanation: On opening SYSPRINT, the blocksize definition, either on the DD card or in the data-set label, was not of the form 4+N*125.

System Action: Compilation is terminated

Programmer Response: Probable user error. Correct the blocksize definition and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM3897I SYSIN DEFINITION IS INVALID

Explanation: On opening SYSIN, the record-format definition, either on the DD card or in the data-set label, was varying. This is invalid.

System Action: Compilation is terminated

Programmer Response: Probable user error. Correct the definition of SYSIN and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

W IEM3898I COMPILER CORE REQUIREMENT EXCEEDED SIZE GIVEN. AUXILIARY STORAGE USED.

System Action: SPILL file opened

W IEM3899I A BLOCK FOR OVERFLOW DICTIONARY ENTRY OFFSETS WAS CREATED DURING COMPILER PHASE YY

Explanation: This message occurs only in compilations run with the extended dictionary option. An entry offset table in a dictionary block became full before the entries filled the block.

System Action: The block is created to hold the entry offsets overflowing from any entry offset tables during this compilation.

E IEM3900I ERROR IN PROCESS STATEMENT.

Explanation: This message is written directly on SYSPRINT. The syntax of the PROCESS statement is incorrect.

System Action: An attempt is made to interpret the statement correctly. Actual results will depend on the nature of the syntax error.

Programmer Response: Probable user error. Check that the options required have been correctly applied. If not, and recompilation is necessary, correct the syntax of the PROCESS statement. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

E IEM3901I ERROR IN PROCESS STATEMENT. DEFAULT OPTIONS ASSUMED.

Explanation: This message is written directly on SYSPRINT. Invalid syntax in the PROCESS statement has rendered the options unrecognizable.

System Action: The installation defaults are assumed for all options.

Programmer Response: Probable user error. If the use of installation default options is unsatisfactory, correct the syntax of the PROCESS statement and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

E IEM3902I OBJNM FIELD TOO LARGE. FIRST EIGHT CHARACTERS OF NAME HAVE BEEN USED.

Explanation: The name specified in the OBJNM option may not have more than eight characters.

System Action: First eight characters of name used.

Programmer Response: Probable user error. Either amend object module name as required, or alter other references to object module to correspond with truncated name. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options

'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

W IEM3903I CARRIAGE CONTROL POSITION LIES WITHIN THE SOURCE MARGIN. IT HAS BEEN IGNORED.

Programmer Response: Probable user error. Recompile with carriage control position outside source margin. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

E IEM3904I THE FOLLOWING STRING NOT IDENTIFIED AS A KEYWORD - yyyy

Explanation: This message is written directly on SYSPRINT. The compiler was processing the option list passed to it as an invocation parameter, when it found a character string that it could not identify as a keyword.

System Action: The offending character string is ignored.

Programmer Response: Probable user error. Correct the erroneous parameter, and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

E IEM3905I THE FOLLOWING KEYWORD DELETED, DEFAULT USED FOR - yyyy

Explanation: This message is written directly on SYSPRINT. The compiler was processing the option list passed to it as an invocation parameter, when it found an option keyword that had been deleted at system generation.

System Action: The keyword passed at invocation time is ignored. The default interpretation for the option, as set at system generation, is used.

Programmer Response: Probable user error. None, unless it is required to reinstate the deleted keyword, in which case it is necessary to generate the required version of the compiler with a system generation run. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.

- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.

- Have the associated job stream and source program listing available.

E IEM3906I OPTION SPECIFICATION CONTAINS INVALID SYNTAX, DEFAULT USED FOR - yyyy

Explanation: This message is written directly on SYSPRINT. The compiler was processing the option list passed to it as an invocation parameter, when it found that a sub-parameter, associated with the keyword given in the diagnostic message, was incorrectly specified.

System Action: The keyword passed at invocation time is ignored. The default interpretation for the option, as set at system generation, is used.

Programmer Response: Probable user error. Correct the erroneous parameter, and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.

- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.

- Have the associated job stream and source program listing available.

E IEM3907I THE FOLLOWING NAME IGNORED AS IT DOES NOT APPEAR IN THE PHASE DIRECTORY - yy

Explanation: This message is written directly on SYSPRINT. The two characters given in the message were used as parameters to the DUMP option. This usage is incorrect since the characters do not represent the name of a compiler phase.

System Action: The processing of the DUMP option continues, unless the two characters were used to indicate the first phase of an inclusive phase dump, in which case the scan of the DUMP option is terminated.

Programmer Response: Probable user error. Correct the erroneous parameter, and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.

- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.

- Have the associated job stream and source program listing available.

S IEM3908I SYNTAX ERROR IN DUMP OPTION SPECIFICATION

Explanation: This message is

written directly on SYSPRINT. Incorrect use of delimiters in the specification of the DUMP option parameters.

System Action: Processing of DUMP option is terminated

Programmer Response: Probable user error. Correct the erroneous specification, and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM3909I EXTENDED DICTIONARY CAPACITY EXCEEDED. COMPILATION TERMINATED.

Explanation: This message occurs only in compilations run with the extended dictionary option. The block created to hold overflow dictionary entry offsets is full.

System Action: Compilation is terminated

Programmer Response: Probable user error. Subdivide program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM3910I SYSPRINT BLOCKSIZE IS TOO LARGE WITH THIS SIZE OPTION

Explanation: The size specified allows a limited buffer area which is smaller than that required by the specified blocksize.

System Action: Compilation is terminated

Programmer Response: Probable user error. Use smaller blocksize or larger SIZE option. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

W IEM3911I SIZE AVAILABLE FOUND TO BE yyyyyy BYTES. SIZE=44K ASSUMED. COMPILATION CONTINUES.

Explanation: SIZE is found to be less than 44K.

Programmer Response: Probable

user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM3912I SYSIN BLOCKSIZE IS TOO LARGE WITH THIS SIZE OPTION

Explanation: The size specified allows a limited buffer area which is smaller than that required by the buffers for SYSIN, or for SYSIN and SYSPRINT together.

System Action: Compilation is terminated

Programmer Response: Probable user error. Ensure that SIZE option allows room for both the SYSIN and the SYSPRINT buffers. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

S IEM3913I SYSPUNCH BLOCKSIZE IS TOO LARGE WITH THIS SIZE OPTION. THE DECK AND MACDCK OPTIONS HAVE BEEN DELETED.

Explanation: The SIZE specified allows a limited buffer area which is smaller than that required by the specified SYSPUNCH blocksize.

System Action: The DECK and MACDCK options are deleted

Programmer Response: Probable user error. Ensure that the SIZE option allows room for the SYSPUNCH buffers needed, and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

S IEM3914I SYSLIN BLOCKSIZE IS TOO LARGE WITH THIS SIZE OPTION. THE LOAD OPTION HAS BEEN DELETED.

Explanation: The SIZE specified allows a limited buffer area which is smaller than that required by the specified SYSLIN blocksize.

System Action: The LOAD option is deleted.

Programmer Response: Probable

user error. Ensure that the SIZE option allows room for the SYSLIN buffers needed and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

E IEM3915I THE CONFLICTING COMPILER OPTIONS MACDCK AND NOMACRO HAVE BEEN SPECIFIED. THE MACDCK OPTION HAS BEEN DELETED.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

E IEM3916I THE CONFLICTING COMPILER OPTIONS DECK AND NOCOMP HAVE BEEN SPECIFIED. THE DECK OPTION HAS BEEN DELETED.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

E IEM3917I THE CONFLICTING COMPILER OPTIONS LOAD AND NOCOMP HAVE BEEN SPECIFIED. THE LOAD OPTION HAS BEEN DELETED.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

Compile-Time Processing Diagnostic Messages

The details given under the heading "Source Program Diagnostic Messages" apply equally to compile-time processing messages, with one exception: all compile-time processing messages are listed in a group following the SOURCE2 input listing and preceding the source program listing.

The line number in the messages refers to the line in which the error was found. The incorrect statement may have commenced on an earlier line.

S IEM4106I UNEXPECTED END-OF-FILE IN STRING AT OR BEYOND LINE NUMBER xxx. A STRING DELIMITER HAS BEEN INSERTED.

Explanation: End-of-file encountered while scanning for closing quote of a string constant.

System Action: Closing quote inserted before end-of-file.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM4109I REPLACEMENT VALUE IN LINE NUMBER xxx CONTAINS UNDELIMITED STRING. PROCESSING TERMINATED.

Explanation: End-of-string delimiter cannot be found in a replacement value.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM4112I ILLEGAL CHARACTER IN APPARENT BIT STRING IN LINE NUMBER xxx. STRING TREATED AS A CHARACTER STRING.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4115I UNEXPECTED END-OF-FILE IN COMMENT AT OR BEYOND LINE NUMBER xxx. A COMMENT DELIMITER HAS BEEN INSERTED.

Explanation: End-of-file encountered while scanning for end-of-comment delimiter.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM4118I REPLACEMENT VALUE IN LINE NUMBER xxx CONTAINS UNDELIMITED COMMENT. PROCESSING TERMINATED.

Explanation: End-of-comment delimiter cannot be found in a replacement value.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM4121I INVALID CHARACTER HAS BEEN REPLACED BY BLANK IN OR FOLLOWING LINE NUMBER xxx

Explanation: Invalid character found in source text

Programmer Response: Probable user error. Correct program

and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM4124I COMPILER ERROR. PUSH DOWN STACK OUT OF PHASE

System Action: Processing terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM4130I UNDECLARED IDENTIFIER zzzz REFERENCED AT LINE NUMBER xxx. PROCESSING TERMINATED.

Explanation: An attempt is made to execute a statement which references an identifier for which a DECLARE statement has not been executed.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4133I % ENCOUNTERED IN LABELLIST OF STATEMENT IN LINE NUMBER xxx. IT HAS BEEN IGNORED.

Programmer Response: Probable user error. Remove % from label list. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4134I UNEXPECTED COLON WITHOUT PRECEDING LABEL IN LINE NUMBER xxx. COLON HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4136I STATEMENT TYPE NOT RECOGNIZABLE IN LINE NUMBER xxx. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4139I PREVIOUS USAGE OF IDENTIFIER zzzz CONFLICTS WITH USE AS LABEL IN LINE NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless an attempt is made to execute a statement which references the ill-defined identifier.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4142I LABEL zzzz IN LINE NUMBER xxx MULTIPLY DEFINED. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless a statement which references the multiply defined label is executed.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM4143I LABELS BEFORE DECLARE STATEMENT IN LINE NUMBER xxx ARE IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4148I IDENTIFIER zzzz IN LINE NUMBER xxx USED WITH CONFLICTING ATTRIBUTES. ANY REFERENCE WILL TERMINATE PROCESSING.

Explanation: Usage of identifier conflicts with a previous usage or declaration. If the line number refers to a procedure END statement, the error occurred within the procedure.

System Action: No action unless a statement is executed which references the identifier in error.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4150I FORMAL PARAMETER zzzz WAS NOT DECLARED IN PROCEDURE ENDING IN LINE NUMBER xxx. TYPE CHARACTER HAS BEEN FORCED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4151I LABEL zzzz IS NOT DEFINED. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless a statement is executed which references the undefined label.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4152I END OF FILE OCCURS BEFORE END FOR CURRENT PROCEDURE OR DO. END HAS BEEN INSERTED AT LINE NUMBER xxx.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4153I LABEL zzzz IS UNDEFINED IN THE PROCEDURE ENDING IN LINE NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

Explanation: Label may have been defined outside of procedure, but transfers out of procedures are not allowed.

System Action: Any reference to the label in the procedure will terminate processing.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4154I SEMICOLON TERMINATES IF EXPRESSION IN LINE NUMBER xxx. SEMICOLON HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4157I NEITHER % NOR THEN FOLLOWS IF EXPRESSION IN LINE NUMBER xxx. IF STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4160I % MISSING BEFORE THEN OF IF STATEMENT IN LINE NUMBER xxx. % HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4163I THEN MISSING FOLLOWING % IN IF STATEMENT IN LINE NUMBER xxx. A THEN HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4166I COMPILE TIME STATEMENT MUST FOLLOW THEN OR ELSE IN LINE NUMBER xxx. A % HAS BEEN INSERTED IN FRONT OF STATEMENT.

Explanation: % does not precede the first statement in the THEN or ELSE clause of an IF statement.

Programmer Response: Probable user error. If the statement in question is meant to be a non-compile time statement, it should be put inside of a "% DO" group. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4169I THEN MISSING FROM IF STATEMENT AT LINE NUMBER xxx IN A COMPILE TIME PROCEDURE. A THEN HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4172I THE % IN LINE NUMBER xxx IS NOT ALLOWED IN COMPILE TIME PROCEDURES. IT HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM4175I LABELS BEFORE ELSE IN LINE NUMBER xxx HAVE BEEN IGNORED.

Explanation: Label(s) found preceding an ELSE statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4176I NO STATEMENT FOLLOWS THEN OR ELSE IN LINE NUMBER xxx. A NULL STATEMENT HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4178I ELSE WITHOUT PRECEDING IF IN LINE NUMBER xxx HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4184I ASSIGNMENT STATEMENT IN LINE NUMBER xxx MUST END WITH SEMICOLON. TEXT DELETED TILL SEMICOLON IS FOUND.

Programmer Response: Probable user error. Correct program and recompile. If the problem

recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4187I LABEL MISSING FROM PROCEDURE STATEMENT IN LINE NUMBER xxx. A DUMMY LABEL HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM4188I IMPLEMENTATION RESTRICTION. NO MORE THAN 254 COMPILE-TIME PROCEDURES MAY BE DEFINED IN A COMPILATION. PROCESSING TERMINATED.

Programmer Response: Probable user error. Delete excess procedures. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM4190I LABEL zzzz ON PROCEDURE IN LINE NUMBEER xxx IS PREVIOUSLY DEFINED. ANY REFERENCE TO IT WILL TERMINATE PROCESSING.

System Action: No action unless a statement is executed which references the multiply defined label.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4193I ILLEGAL USE OF FUNCTION NAME zzzz ON LEFT HAND SIDE OF EQUALS SYMBOL. ANY REFERENCE WILL TERMINATE PROCESSING.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4196I PREVIOUS USE OF IDENTIFIER zzzz CONFLICTS WITH USE AS ENTRY NAME IN LINE NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless a statement is executed which references the erroneous identifier.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4199I FORMAL PARAMETER zzzz IS REPEATED IN PARAMETER LIST IN LINE NUMBER xxx. THE SECOND OCCURRENCE HAS BEEN REPLACED BY A DUMMY PARAMETER.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4202I IMPLEMENTATION RESTRICTION: MORE THAN 15 PARAMETERS OCCUR IN LINE NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: Processing is terminated if an attempt is made to execute a statement which references the procedure that has more than 15 parameters.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before

calling IBM for programming support:

- Have the source program listing available.

E IEM4205I FORMAL PARAMETER MISSING IN LINE NUMBER xxx. A DUMMY HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4208I UNRECOGNIZABLE PARAMETER yyyy IN LINE NUMBER xxx. IT HAS BEEN REPLACED BY A DUMMY PARAMETER.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4211I PARAMETER IN LINE NUMBER xxx NOT FOLLOWED BY COMMA OR PARENTHESIS. TEXT DELETED TO NEXT COMMA OR END OF STATEMENT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4212I UNEXPECTED END OF PROCEDURE STATEMENT IN LINE NUMBER xxx. RIGHT PARENTHESIS INSERTED.

Explanation: A semicolon was encountered during scan of an apparent parameter list.

System Action: A right parenthesis is inserted before the semicolon and processing continues.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4214I ILLEGAL FORM OF RETURNS OPTION IN LINE NUMBER xxx. RETURNS (CHAR) HAS BEEN ASSUMED.

Explanation: RETURNS option should be of the form RETURNS(CHAR|FIXED).

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4215I DATA ATTRIBUTE IN PROCEDURE STATEMENT IN LINE NUMBER xxx IS NOT PARENTHESIZED AND IS NOT PRECEDED BY RETURNS. RETURNS AND PARENTHESES HAVE BEEN ASSUMED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4216I THERE IS NO RIGHT PARENTHESIS FOLLOWING THE DATA ATTRIBUTE OF THE RETURNED VALUE IN LINE NUMBER xxx. ONE HAS BEEN ASSUMED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4217I NO ATTRIBUTE FOR RETURNED VALUE IN LINE NUMBER xxx. CHARACTER ATTRIBUTE IS USED.

System Action: CHARACTER attribute is assigned

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before

calling IBM for programming support:

- Have the source program listing available.

S IEM4220I SEMICOLON NOT FOUND WHERE EXPECTED IN PROCEDURE STATEMENT IN LINE NUMBER xxx. TEXT DELETED UP TO NEXT SEMICOLON.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4223I ENTRY ATTRIBUTE AND PROCEDURE STATEMENT FOR ENTRY zzzz DISAGREE ON THE NUMBER OF PARAMETERS. THE LATTER IS USED.

System Action: The number of parameters specified in the PROCEDURE statement is used.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4226I RETURNS ATTRIBUTE AND PROCEDURE STATEMENT FOR ENTRY zzzz DISAGREE ON ATTRIBUTE OF RETURNED VALUE.

System Action: The returned value will first be converted to the type on the procedure statement and will then be converted to the type given in the RETURNS attribute. A third conversion can occur if the type given in the returns attribute does not agree with the type required where the result is used.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4229I PROCEDURE STATEMENT AT LINE NUMBER xxx MAY NOT BE USED WITHIN A PROCEDURE. PROCEDURE HAS BEEN DELETED.

Explanation: Compile-time procedures may not be nested.

System Action: Text is deleted up to and including the first % END following the erroneous PROCEDURE statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4232I PROCEDURE STATEMENT AT LINE NUMBER xxx MAY NOT FOLLOW THEN OR ELSE. PROCEDURE HAS BEEN REPLACED BY A NULL STATEMENT.

Explanation: A PROCEDURE statement may appear in a THEN or ELSE clause only if it is inside a compile-time DO group.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4235I RETURN STATEMENT IN LINE NUMBER xxx IS NOT ALLOWED OUTSIDE OF COMPILE-TIME PROCEDURE. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4238I RETURNED VALUE MUST BE PARENTHESIZED IN LINE NUMBER xxx. PARENTHESIS INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4241I RETURNS EXPRESSION IN LINE NUMBER xxx DOES NOT END RETURN STATEMENT. REMAINDER OF STATEMENT HAS BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4244I GOTO IN LINE NUMBER xxx IS NOT FOLLOWED BY LABEL. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4247I PREVIOUS USE OF IDENTIFIER zzzz CONFLICTS WITH USE AS OBJECT OF GOTO IN LINE NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless a statement is executed which references the erroneous identifier.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4248I SEMICOLON NOT FOUND WHERE EXPECTED IN GOTO STATEMENT IN LINE NUMBER xxx. TEXT DELETED UP TO NEXT SEMICOLON.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM4250I GOTO zzzz IN LINE NUMBER xxx TRANSFERS CONTROL INTO ITERATIVE DO OR ENCLOSED INCLUDED TEXT. PROCESSING TERMINATED.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM4253I ACTIVATE OR DEACTIVATE IN LINE NUMBER xxx NOT ALLOWED IN A COMPILE-TIME PROCEDURE. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4254I EMPTY ACTIVATE OR DEACTIVATE STATEMENT IN LINE NUMBER xxx. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4256I SURPLUS COMMA IN ACTIVATE OR DEACTIVATE IN LINE NUMBER xxx. THE COMMA HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4259I UNRECOGNIZABLE FIELD IN

ACTIVATE OR DEACTIVATE STATEMENT IN LINE NUMBER xxx. THE FIELD HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4262I ONLY PROCEDURES OR VARIABLES MAY HAVE ACTIVITY CHANGED. IDENTIFIER zzzz IN LINE NUMBER xxx HAS BEEN DELETED FROM STATEMENT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4265I COMMA MUST SEPARATE FIELDS OF ACTIVATE AND DEACTIVATE STATEMENTS. IN LINE NUMBER xxx TEXT AFTER IDENTIFIER yyyy HAS BEEN DELETED UP TO NEXT COMMA.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4271I INVALID SYNTAX IN DO STATEMENT IN LINE NUMBER xxx. IT HAS BEEN CONVERTED TO A GROUPING DO.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM4277I NO MAXIMUM VALUE WAS SPECIFIED IN ITERATIVE DO IN LINE NUMBER xxx. PROGRAM WILL LOOP UNLESS ALTERNATE EXIT IS PROVIDED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4280I UNEXPECTED % IN LINE NUMBER xxx TREATED AS HAVING BEEN PRECEDED BY SEMICOLON.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4283I MULTIPLE TO'S HAVE OCCURRED IN DO STATEMENT IN LINE NUMBER xxx. SECOND 'TO' HAS BEEN CHANGED TO 'BY'.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4286I MULTIPLE BY'S HAVE OCCURRED IN DO STATEMENT IN LINE NUMBER xxx. SECOND 'BY' HAS BEEN CHANGED TO 'TO'.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4289I DO STATEMENT IN LINE NUMBER xxx SHOULD END WITH SEMICOLON. TEXT TO SEMICOLON DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4292I END STATEMENT AT LINE NUMBER xxx MAY NOT FOLLOW THEN OR

ELSE. A NULL STATEMENT HAS BEEN INSERTED BEFORE THE END STATEMENT.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4295I SEMICOLON NOT FOUND WHERE EXPECTED IN END STATEMENT IN LINE NUMBER xxx. TEXT DELETED UP TO SEMICOLON.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4296I END STATEMENT IN LINE NUMBER xxx NOT PRECEDED BY DO OR PROCEDURE STATEMENT. END HAS BEEN DELETED.

Explanation: An END statement has been encountered which is not preceded by a DO or PROCEDURE statement that has not already been terminated.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4298I LABEL REFERENCED ON END STATEMENT IN LINE NUMBER xxx CANNOT BE FOUND. END TREATED AS HAVING NO OPERAND.

Explanation: The label cannot be found on a DO or PROCEDURE statement that has not already been terminated.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4299I END STATEMENT ENDING PROCEDURE IN LINE NUMBER xxx DID NOT HAVE A PRECEDING PERCENT. A PERCENT IS INSERTED.

Explanation: The END statement referred to in this message is the logical end of the procedure.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4301I IDENTIFIER zzzz ON END STATEMENT IN LINE NUMBER xxx IS NOT A LABEL. END TREATED AS HAVING NO OPERAND.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4304I PROCEDURE zzzz DID NOT INCLUDE A RETURN STATEMENT.

Explanation: Language syntax requires use of RETURN statement in a procedure.

System Action: A null value is returned if the procedure is invoked.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4307I INCLUDE STATEMENT AT LINE NUMBER xxx IS NOT ALLOWED IN COMPILE-TIME PROCEDURES. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before

calling IBM for programming support:

- Have the source program listing available.

E IEM4310I IMPLEMENTATION RESTRICTION. DDNAME IN LINE NUMBER xxx HAS BEEN TRUNCATED TO 8 CHARACTERS.

Explanation: The first of a pair of data set identifiers in an INCLUDE statement is a ddname and as such is limited to a maximum of 8 characters.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4313I UNRECOGNIZABLE FIELD IN INCLUDE STATEMENT AT LINE NUMBER xxx. FIELD HAS BEEN DELETED.

System Action: Text is deleted up to next comma or semicolon.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4319I EMPTY INCLUDE STATEMENT IN LINE NUMBER xxx. STATEMENT DELETED.

Explanation: At least one identifier must appear in an INCLUDE statement i.e., the data set member name.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4322I IMPLEMENTATION RESTRICTION. MEMBER NAME IN LINE NUMBER xxx HAS BEEN TRUNCATED TO 8 CHARACTERS.

System Action: First 8 characters of member name have been used.

Programmer Response: Probable user error. Correct data set member name in INCLUDE statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4325I RIGHT PARENTHESIS INSERTED AFTER MEMBER NAME IN LINE NUMBER xxx.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4326I LEFT PARENTHESIS INSERTED BEFORE MEMBER NAME IN LINE NUMBER xxx.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM4328I COMPILER ERROR. DICTIONARY INFORMATION INCORRECT.

Explanation: A name containing an invalid character is found in the dictionary.

System Action: Processing is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM4331I DECLARE STATEMENT IN LINE NUMBER xxx IS ILLEGAL AFTER THEN OR ELSE. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program. A DECLARE statement can appear in the THEN or ELSE clause of an IF statement if it is inside a DO group. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4332I EMPTY DECLARE STATEMENT IN LINE NUMBER xxx. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4334I IMPLEMENTATION RESTRICTION. FACTORING IN DECLARE STATEMENT IN LINE NUMBER xxx EXCEEDS 3 LEVELS. REMAINDER OF STATEMENT DELETED.

Programmer Response: Probable user error. Reduce level of factoring in DECLARE statement. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4337I SURPLUS COMMA HAS BEEN FOUND IN DECLARE STATEMENT IN LINE NUMBER xxx. THIS COMMA HAS BEEN DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4340I IDENTIFIER MISSING WHERE EXPECTED IN LINE NUMBER xxx. A DUMMY IDENTIFIER HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4343I IDENTIFIER zzzz IN LINE NUMBER xxx HAS MULTIPLE DECLARATIONS. ANY REFERENCE WILL TERMINATE PROCESSING.

Explanation: An identifier may be declared only once.

System Action: No action unless a statement is executed which references the multiply declared identifier.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4346I UNRECOGNIZABLE SYNTAX IN DECLARE STATEMENT IN LINE NUMBER xxx. STATEMENT DELETED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4349I LABEL zzzz CANNOT BE DECLARED IN LINE NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

Explanation: An attempt has been made to declare an identifier which has already been used as a label.

System Action: No action unless a statement is executed which references the declared label.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4352I EXTRA PARENTHESIS DELETED IN LINE NUMBER xxx.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4355I ILLEGAL ATTRIBUTE yyyy IN LINE NUMBER xxx. ATTRIBUTE HAS BEEN DELETED.

Explanation: Legal attributes are FIXED, CHARACTER, ENTRY and RETURNS.

System Action: The illegal attribute is deleted.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4358I CLOSING RIGHT PARENTHESIS INSERTED IN LINE NUMBER xxx.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4361I RETURNS ATTRIBUTE OCCURRED WITHOUT ENTRY ATTRIBUTE FOR PROCEDURE zzzz IN DECLARE STATEMENT AT OR BEFORE LINE NUMBER xxx.

Explanation: Both ENTRY and RETURNS attributes must be declared for a compile-time procedure name.

System Action: The identifier is treated as an ENTRY name. If it is referenced, the arguments will be converted to the types declared for the procedure parameters.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4364I NO ATTRIBUTES WERE DECLARED FOR IDENTIFIER zzzz IN DECLARE STATEMENT AT OR BEFORE LINE NUMBER xxx. CHARACTER HAS BEEN ASSIGNED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4367I RETURNS ATTRIBUTE NOT GIVEN FOR ENTRY NAME zzzz IN DECLARE STATEMENT AT OR BEFORE LINE NUMBER xxx.

Explanation: Both ENTRY and RETURNS attributes must be declared for a compile-time procedure name.

System Action: The attribute of the returned value is determined by the relevant PROCEDURE statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4370I ENTRY ATTRIBUTE DISAGREES WITH DECLARATION FOR FORMAL PARAMETER zzzz. THE LATTER HAS BEEN USED.

Explanation: An ENTRY attribute in a DECLARE statement does not agree with the parameter attributes declared in the procedure.

System Action: If the relevant procedure is referenced, the argument will be converted to the type declared for the formal parameter.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4373I RETURNS ATTRIBUTE IN LINE NUMBER xxx MUST BE PARENTHESIZED. PARENTHESIS INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4376I ONLY FIXED OR CHARACTER ARE ALLOWED IN RETURNS ATTRIBUTE IN LINE NUMBER xxx. ATTRIBUTE IGNORED.

Explanation: An illegal attribute was found.

System Action: The attribute of the returned value is determined by the relevant PROCEDURE statement.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4379I ATTRIBUTE yyyy IS ILLEGAL IN ENTRY ATTRIBUTE IN LINE NUMBER xxx. NO CONVERSION WILL BE DONE.

Explanation: An invalid attribute was found.

System Action: No conversion to an ENTRY attribute will be carried out. However, if the relevant procedure is referenced, arguments will be converted to the types declared for the procedure parameters.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4382I ATTRIBUTE CONFLICTS WITH PREVIOUS ATTRIBUTE FOR IDENTIFIER zzzz IN LINE NUMBER xxx. ATTRIBUTE IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4383I PREVIOUS USAGE OF IDENFIFIER zzzz CONFLICTS WITH ATTRIBUTE IN LINE NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4391I OPERAND MISSING IN LINE NUMBER xxx. A FIXED DECIMAL ZERO HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4394I ILLEGAL OPERATOR yyyy IN LINE NUMBER xxx. IT HAS BEEN REPLACED BY A PLUS.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM4397I A LETTER IMMEDIATELY FOLLOWS CONSTANT yyyy IN LINE NUMBER

xxx. AN INTERVENING BLANK HAS BEEN ASSUMED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4400I OPERATOR .NOT. IN LINE NUMBER xxx USED AS AN INFIX OPERATOR. IT HAS BEEN REPLACED BY .NE.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM4403I COMPILER ERROR. EXPRESSION SCAN OUT OF PHASE.

System Action: Processing is terminated.

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM4406I PREVIOUS USAGE OF IDENTIFIER zzzz CONFLICTS WITH USE IN EXPRESSION IN LINE NUMBER xxx.

System Action: Processing is terminated if an attempt is made to execute a statement which references the identifier in question.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4407I UNDECIPHERABLE KEYWORD. nnn IDENTIFIERS HAVE BEEN DELETED BEFORE yyyy IN LINE NUMBER xxx.

Explanation: The processor has found a mis-match while scanning a keyword consisting of more than one identifier.

System Action: The identifiers preceding the non-matching identifier are deleted.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4409I OPERATOR MISSING IN LINE NUMBER xxx. A PLUS HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4412I NO EXPRESSION WHERE ONE IS EXPECTED IN LINE NUMBER xxx. A FIXED DECIMAL ZERO HAS BEEN INSERTED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4415I ILLEGAL OPERAND yyyy IN LINE NUMBER xxx HAS BEEN REPLACED BY A FIXED DECIMAL ZERO.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4421I MISSING LEFT PARENTHESIS INSERTED AT BEGINNING OF EXPRESSION IN LINE NUMBER xxx.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM4433I REFERENCE IN LINE NUMBER xxx TO STATEMENT OR IDENTIFIER WHICH IS IN ERROR. PROCESSING TERMINATED.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

S IEM4436I EXCESS ARGUMENTS TO FUNCTION zzzz IN LINE NUMBER xxx. EXTRA ARGUMENTS HAVE BEEN DELETED.

Explanation: Too many arguments appear in a procedure reference.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

W IEM4439I TOO FEW ARGUMENTS TO FUNCTION zzzz IN LINE NUMBER xxx. MISSING ARGUMENTS HAVE BEEN REPLACED BY NULL STRINGS OR FIXED DECIMAL ZEROS.

Explanation: Too few arguments appear in a procedure reference.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4448I NO ENTRY DECLARATION FOR PROCEDURE zzzz REFERENCED IN LINE NUMBER xxx. ATTRIBUTES TAKEN FROM PROCEDURE.

Explanation: All procedure names must be declared with ENTRY and RETURNS attributes before the procedure is referenced.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM4451I PROCEDURE zzzz REFERENCED IN LINE NUMBER xxx CANNOT BE FOUND. PROCESSING TERMINATED.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM4452I RECURSIVE USE OF PROCEDURE zzzz IN LINE NUMBER xxx IS DISALLOWED. PROCESSING TERMINATED.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

E IEM4454I TOO FEW ARGUMENTS HAVE BEEN SPECIFIED FOR THE BUILTIN FUNCTION SUBSTR IN LINE NUMBER xxx. A NULL STRING HAS BEEN RETURNED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4457I TOO MANY ARGUMENTS HAVE BEEN SPECIFIED FOR THE BUILTIN FUNCTION SUBSTR IN LINE NUMBER xxx. EXTRA ARGUMENTS HAVE BEEN IGNORED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4460I FIXED OVERFLOW HAS OCCURRED IN LINE NUMBER xxx. RESULT TRUNCATED.

System Action: Truncation occurs on left to 5 decimal digits.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4463I ZERO DIVIDE HAS OCCURRED AT LINE NUMBER xxx. RESULT SET TO ONE.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4469I END-OF-FILE FOUND IMBEDDED IN STATEMENT IN LINE NUMBER xxx. EXECUTION OF STATEMENT WILL CAUSE TERMINATION.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4472I IDENTIFIER BEGINNING zzzz IN STATEMENT AT LINE NUMBER xxx IS TOO LONG AND HAS BEEN TRUNCATED.

Explanation: Identifiers may not exceed 31 characters in length.

System Action: The identifier is truncated to the first 31 characters.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4473I CONSTANT yyyy IN LINE NUMBER xxx HAS PRECISION GREATER THAN 5. A FIXED DECIMAL ZERO HAS BEEN INSERTED.

Explanation: Implementation restriction. Precision of fixed decimal numbers is limited to 5 digits.

System Action: A value of zero is assigned.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4475I QUESTION MARK IN LINE NUMBER xxx HAS NO SIGNIFICANCE. IT HAS BEEN IGNORED

Explanation: Question mark, although a recognizable character in PL/I, has no syntactical meaning.

Programmer Response: Probable user error. Correct program and recompile. If the problem

recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM4478I STRING IN LINE NUMBER xxx CONVERTS TO A FIXED DECIMAL NUMBER WITH PRECISION GREATER THAN 5. PROCESSING TERMINATED.

Explanation: Implementation restriction. Precision of fixed decimal numbers is limited to 5 digits.

System Action: Processing is terminated

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM4481I CHARACTER STRING IN LINE NUMBER xxx CONTAINS CHARACTER OTHER THAN 1 OR 0 AND CANNOT BE CONVERTED TO A BIT STRING. PROCESSING TERMINATED.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM4484I STRING IN LINE NUMBER xxx OR IN PROCEDURE REFERENCED IN SAID LINE NUMBER CANNOT BE CONVERTED TO A FIXED DECIMAL CONSTANT. PROCESSING TERMINATED.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM4499I A % STATEMENT IS FOUND IN A REPLACEMENT VALUE IN LINE NUMBER xxx. PROCESSING TERMINATED.

Explanation: A replacement value may not contain a compile-time statement.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM4502I AN IDENTIFIER zzzz WITH CONFLICTING USAGE OR MULTIPLE DEFINITIONS IS REFERENCED IN LINE NUMBER xxx. PROCESSING TERMINATED.

Explanation: An attempt is made to execute a statement which references an identifier that was not properly defined.

Programmer Response: Probable user error. Correct program. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the

comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

S IEM4504I VARIABLE zzzz IS USED IN LINE NUMBER xxx BEFORE IT IS INITIALIZED. IT HAS BEEN GIVEN NULL STRING OR ZERO VALUE.

Explanation: A value must be assigned to variables before they are referenced after being declared.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM4505I DD STATEMENT FOR INCLUDE zzzz MISSING IN LINE NUMBER xxx. PROCESSING TERMINATED.

Explanation: A DD statement must be present, in the Job Control cards for the compilation, with a name in the name field that corresponds to the ddname identifier in the INCLUDE statement. If no ddname is specified in the INCLUDE statement, a SYSLIB DD statement is required.

Programmer Response: Probable user error. Insert appropriate DD statement and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM4508I UNRECOVERABLE I/O ERROR WHILE SEARCHING FOR MEMBER OF INCLUDE zzzz IN LINE NUMBER xxx. PROCESSING TERMINATED.

Programmer Response: Check DD statement and reattempt compilation. If the input/output error persits, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement was included for the failing job step.
- Have the associated job stream and source program listing available.

T IEM4511I ILLEGAL RECORD FORMAT SPECIFIED FOR INCLUDE zzzz IN LINE NUMBER xxx. PROCESSING TERMINATED.

Explanation: Included records must be a fixed length of not more than 100 characters with a maximum blocking factor of 5. Blocksize must be a multiple of the record length.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.

- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.

- Have the associated job stream and source program listing available.

T IEM4514I MEMBER OF INCLUDE zzzz IN LINE NUMBER xxx NOT FOUND ON DATA SET. PROCESSING TERMINATED.

Programmer Response: Probable user error. Check INCLUDE statement, DD statement and data file. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.

- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.

- Have the associated job stream and source program listing available.

W IEM4517I RECORD LENGTH NOT SPECIFIED FOR INCLUDE zzzz IN LINE NUMBER xxx. RECORD LENGTH EQUAL TO BLOCKSIZE HAS BEEN ASSUMED.

Programmer Response: Probable user error. Correct record length specification in DD statement, if necessary. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.

- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.

- Have the associated job stream and source program listing available.

W IEM4520I BLOCKSIZE NOT SPECIFIED FOR INCLUDE zzzz IN LINE NUMBER xxx. BLOCKSIZE EQUAL TO RECORD LENGTH HAS BEEN ASSUMED.

Programmer Response: Probable user error. Correct blocksize specification in DD statement, if necessary. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.

- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.

- Have the associated job stream and source program listing available.

W IEM4523I RECORD LENGTH AND BLOCKSIZE NOT SPECIFIED FOR INCLUDE zzzz IN LINE NUMBER xxx. RECORD LENGTH OF 80 AND BLOCKSIZE OF 400 HAVE BEEN ASSUMED.

Programmer Response: Probable user error. Correct record length and block size specifications in DD statement, if necessary. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a

formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.
- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.
- Have the associated job stream and source program listing available.

T IEM4526I I/O ERROR WHILE READING TEXT INCLUDED FROM zzzz AT LINE NUMBER xxx. PROCESSING TERMINATED.

Programmer Response: Check DD statement and reattempt compilation. If the input/output error persists, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement was included for the failing job step.
- Have the associated job stream and source program listing available.

T IEM4529I IMPLEMENTATION RESTRICTION. EXCESSIVE LEVEL OF NESTING OR REPLACEMENT AT LINE NUMBER xxx. PROCESSING TERMINATED.

Explanation: Level of nesting in this case is calculated by summing the number of current unbalanced left parentheses, the number of current nested DO's, the number of current nested IF's, and the number of current nested replacements. A level of 50 is always acceptable.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Have the associated job stream and source program listing available.

T IEM4532I INPUT RECORD AT LINE NUMBER xxx IS TOO LONG. PROCESSING TERMINATED.

Explanation: Input record contains more than 100 characters.

Programmer Response: If the input/output error persists, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)
- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement was included for the failing job step.
- Have the associated job stream and source program listing available.

T IEM4535I INPUT RECORD CONTAINS FEWER CHARACTERS THAN SORMGIN REQUIRES. PROCESSING TERMINATED.

Explanation: The length of the input record is less than the left margin of the SORMGIN specification.

Programmer Response: Probable user error. Check SORMGIN option on EXEC control card. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSUDUMP DD statement, was included for the failing job step.

- Obtain a listing of module IEMAF (in SYS1.LINKLIB) using IMASPZAP.

- Have the associated job stream and source program listing available.

T IEM4547I COMPILER ERROR. INSUFFICIENT SPACE FOR TABLES.

System Action: Processing is terminated

Programmer Response: Do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

E IEM4550I RIGHT PARENTHESIS INSERTED IN LINE NUMBER xxx TO END ARGUMENT LIST FOR PROCEDURE zzzz.

Explanation: The argument list referred to is in a source program reference to a compile-time procedure.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

T IEM4553I IN LINE NUMBER xxx ARGUMENT LIST FOR PROCEDURE zzzz CONTAINS COMPILE TIME CODE. PROCESSING TERMINATED.

Explanation: Compile-time code may not be embedded in argument list of compile-time procedure reference.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Recompile the program with compiler options 'S,DP=(PIE,ZZ)' to obtain a formatted dump of the compiler. (Refer to the comments which precede all the IEMnnnnI messages.)

- Have the associated job stream and source program listing available.

E IEM4559I LEFT PARENTHESIS BEGINNING ARGUMENT LIST OF PROCEDURE zzzz WAS NOT FOUND. PROCEDURE WAS INVOKED AT LINE NUMBER xxx WITHOUT ARGUMENTS.

Explanation: The argument list referred to is in a source program reference to a compile-time procedure.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4562I IDENTIFIER IN LINE NUMBER xxx EXCEEDS 31 CHARACTERS. REPLACEMENT WAS DONE ON TRUNCATED FORM zzzz.

Explanation: A non-compile-time source text identifier consists of more than 31 characters.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4570I THE THIRD ARGUMENT OF BUILT-IN

FUNCTION SUBSTR IS NEGATIVE, IN LINE NUMBER xxx. A NULL STRING HAS BEEN RETURNED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4572I THE THIRD ARGUMENT OF BUILT-IN FUNCTION SUBSTR EXCEEDS THE STRING LENGTH, IN LINE NUMBER xxx. THE SUBSTRING HAS BEEN TRUNCATED AT THE END OF THE ORIGINAL STRING.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4574I THE COMBINED SECOND AND THIRD ARGUMENTS OF BUILT-IN FUNCTION SUBSTR EXCEED THE STRING LENGTH, IN LINE NUMBER xxx. THE SUBSTRING HAS BEEN TRUNCATED AT THE END OF THE ORIGINAL STRING.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4576I THE SECOND ARGUMENT OF BUILT-IN FUNCTION SUBSTR IS LESS THAN ONE, IN LINE NUMBER xxx. ITS VALUE HAS BEEN RESET TO ONE.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

E IEM4578I THE SECOND ARGUMENT OF BUILT-IN FUNCTION SUBSTR EXCEEDS THE STRING LENGTH, IN LINE NUMBER xxx. A NULL STRING HAS BEEN RETURNED.

Programmer Response: Probable user error. Correct program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

S IEM4580I AN UNINITIALISED VARIABLE HAS BEEN FOUND IN A BUILT-IN FUNCTION ARGUMENT LIST, IN STATEMENT NUMBER xxx. A NULL STRING HAS BEEN RETURNED.

Programmer Response: Probable user error. Initialise the variable before invoking the built-in function. If the problem recurs, do the following before calling IBM for programming support:

- Have the source program listing available.

Object-Time Diagnostic Messages

The messages in the following text may be printed on the output data set specified for SYSPRINT, as the result of an exceptional or error condition occurring during the execution of a PL/I program. If the SYSPRINT DD statement is absent, then the object-time messages appear on the operator's console, except for the ON CHECK system action messages and the COPY option output, which will not be produced at all in this case.

Each message number is of the form IHEnnnI, where the code IHE indicates a PL/I library message, and nnn the number of the message. The final character I indicates the informative nature of the message.

Diagnostic messages are printed at execution time for two main reasons:

1. An error occurs for which no specific ON-condition exists in PL/I. A diagnostic message is printed, and the ERROR ON-condition is raised.
2. An ON-condition is raised, by compiled code or by the library, and the action required is system action, for which the language specifies COMMENT as part of the necessary action.

Object time diagnostic messages will take one of the following forms:

1. IHEnnnI FILE name - text AT location message

2. IHEnnnI rtname - text AT location message

3. IHEnnnI text AT location message

   where 'name' is the name of the file associated with the error (given only in I/O diagnostic messages)

   'rtname' is the name of the library routine in which the error occurred (given only for computational subroutines).

   'location message' is either

   OFFSET ± hhhhh FROM ENTRY POINT E1

   or

   OFFSET ± hhhhh FROM ENTRY POINT OF cccc ON-UNIT

   Note: If it is a Model 91 or Model 195 message resulting from an imprecise interrupt, "AT OFFSET..." is replaced by "NEAR OFFSET..." since the instruction causing the interrupt cannot be precisely identified.

If the statement number compiler option has been specified, each message will also contain IN STATEMENT nnnnn prior to AT location message. nnnnn gives the number of the statement in which the condition occurred.

The diagnostic messages for other than ON-type errors are mainly self-explanatory. Explanations in the following lists are given only when the message is not self-explanatory.

To assist error determination, use diagnostic aids during debugging runs:

1. Enable SIZE, SUBSCRIPTRANGE, STRINGRANGE conditions.

2. Do not disable any of the conditions: CONVERSION, FIXEDOVERFLOW, OVERFLOW, UNDERFLOW, ZERODIVIDE.

3. Insert an on-unit for the ERROR condition in the Main Procedure, and include a PL1DUMP DD statement for the failing job step. For example: ON ERROR SNAP CALL IHEDUMP;

4. Recompile program with compiler options 'ST,A,X,L'.

5. Use Linkage Editor options 'LIST,MAP', or Linkage Loader options 'PRINT,MAP'.

6. Specify MSGLEVEL=(1,1) on job statement.

7. Include a SYSABEND DD statement for the failing job step.

Note: If the shared library feature is in use, do the following before calling IBM for programming support:

- Obtain a list of the options specified in the PL1LIB macro used during system generation.
- Obtain a Linkage Editor Map of the resident shared library module IHELTVA.

IHE003I SOURCE PROGRAM ERROR IN STATEMENT nnnnn

This message will always contain a statement number whether or not the compiler option is specified.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated job stream, Linkage Editor/Loader Map and program listing available.

IHE004I INTERRUPT IN ERROR HANDLER - PROGRAM TERMINATED

Explanation: When an unexpected program interrupt occurs during the handling of another program interrupt, it indicates that the program has a disastrous error in it, such as DSA chain out of order, instructions overwritten, or such. The program is abnormally terminated, and the above message is printed out at the console. A dump is

produced with a User Completion Code of 4000.

Programmer Response: After making corrections, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSABEND DD statement was included for the failing job step.
- Have the associated job stream, Linkage Editor/Loader Map and program listing available.

IHE005I PSEUDO-REGISTER VECTOR TOO LONG - PROGRAM NOT EXECUTED

Explanation: This error arises when the sum of the number of procedures, the number of files, and the number of controlled variables exceeds 1000. It causes return to the Supervisor from IHESAP; PL/I program is not entered. The message always appears at the console. A return code of 4004 is generated.

Programmer Response: After making corrections, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSABEND DD statement was included for the failing job step.
- Have the associated job stream, Linkage Editor/Loader Map and program listing available.

IHE006I NO MAIN PROCEDURE. PROGRAM TERMINATED.

Explanation: No external procedure in the program has been given the option MAIN. This message appears at the console. A return code of 4008 is generated.

Programmer Response: After making corrections, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSABEND DD statement was included for the failing job step.
- Have the associated job stream, Linkage Editor/Loader Map and program listing available.

IHE009I IHEDUM*. NO PL1DUMP DD CARD. EXECUTION TERMINATED.

Explanation: Execution has been abnormally terminated with a system dump and a system completion code of (3000 + Return Code (if set))

Programmer Response: After making corrections, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSABEND DD statement was included for the failing job step.
- Have the associated job stream, Linkage Editor/Loader Map and program listing available.

IHE010I PROGRAM ENDED BY OS360. RETURN CODE = hhh (a hexadecimal number).

Explanation: The major task has been terminated abnormally by the operating system. The above message appears on the console.

Programmer Response: After making corrections, recompile the program and execute the job step again. If the problem

recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSABEND DD statement was included for the failing job step.
- Have the associated job stream, Linkage Editor/Loader Map and program listing available.

IHE011I KEY ERROR WHEN CLOSING FILE AT END OF TASK

Explanation: An unresolved key error exists for which no condition can now be raised. The above message appears on the console.

Programmer Response: After making corrections, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSABEND DD statement was included for the failing job step.
- Have the associated job stream, Linkage Editor/Loader Map and program listing available.

IHE012I jjj ABENDED[IN STMT nnnnn] [AT OFFSET xxxxxx] [FROM ENTRY POINT ppp] WITH CC xxx {(SYSTEM)|(USER)}

Explanation: Job jjj was terminated by the operating system with completion code xxx, if (SYSTEM) is printed. (USER) is printed if the termination was due to a user specified ABEND. STMT nnnnn is the statement number, which was the last to be executed before the ABEND took place. xxxxxx is the offset from the entry point specified. Message may appear on SYSPRINT if OPEN and not in ERROR, or on the console.

Programmer Response: After making corrections, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSABEND DD statement was included for the failing job step.
- Have the associated job stream, Linkage Editor/Loader Map and program listing available.

IHE013I NO MAIN STORAGE AVAILABLE FOR PL/I STAE EXIT

Explanation: Insufficient main storage available for the PL/I STAE exit routine to load the routine which analyzes the ABEND termination. Message appears on the console.

Programmer Response: After making corrections, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that a SYSABEND DD statement was included for the failing job step.
- Have the associated job stream, Linkage Editor/Loader Map and program listing available.

I/O Errors

IHE018I FILE name - FILE TYPE NOT SUPPORTED

IHE020I FILE name - ATTEMPT TO READ OUTPUT FILE

IHE021I FILE name - ATTEMPT TO WRITE INPUT FILE

IHE022I GET/PUT STRING EXCEEDS STRING SIZE

Explanation:

For input: programmer has requested more than exists on the input string.
For output: programmer is trying to write more than his output string will hold.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE023I FILE name - OUTPUT TRANSMIT ERROR NOT ACCEPTABLE

Explanation: The ERROR is raised, (i) upon return from a TRANSMIT ON-unit, if the device in error is other than a printer, or (ii) if access to a file by RECORD I/O has been attempted after the TRANSMIT condition has been raised for output.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE024I FILE name - PRINT OPTION/FORMAT ITEM FOR NON-PRINT FILE

Explanation: Attempt to use PAGE, LINE or SKIP ≤ 0 for a non-print file.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE025I DISPLAY - MESSAGE OR REPLY AREA LENGTH ZERO

Explanation: This message appears only if the REPLY option is exercised.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE026I FILE name - DATA DIRECTED INPUT - INVALID ARRAY DATUM

Explanation: Number of subscripts on external medium does not correspond to number of declared subscripts.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE027I GET STRING - UNRECOGNIZABLE DATA NAME

Explanation:

1. GET DATA - name of data item found in string is not known at the time of the GET statement, or
2. GET DATA data list - name of data item found in string is not specified in the list.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the NAME condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated job stream, Linkage Editor/Loader Map and program listing available.

IHE029I FILE name - UNSUPPORTED FILE OPERATION

Explanation: Programmer has executed an I/O statement with an option or verb not applicable to the specified file.

For example:

I/O Option or Verb	File Attribute
READ SET| LOCATE	DIRECT| (SEQUENTIAL UNBUFFERED)
REWRITE (without FROM)| EXCLUSIVE| UNLOCK| (READ NOLOCK)	(SEQUENTIAL INPUT|OUTPUT| UPDATE)| (DIRECT INPUT| OUTPUT)
KEYTO	REGIONAL DIRECT
LINESIZE| PAGESIZE	STREAM (INPUT| UPDATE)

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE030I FILE name - REWRITE/DELETE NOT IMMEDIATELY PRECEDED BY READ

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem

recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE031I FILE name - INEXPLICABLE I/O ERROR

Explanation: Operating system data management has detected some error in the current input/output operation. The message could be caused by one of the following:

1. INDEXED data set with F-format records: a previously created data set was reopened for sequential output and the key of the record to be added was not higher in the collating sequence than that of the last key on the data set.
2. An input/output error occurred for which no information was supplied by data management.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE032I FILE name - OUTSTANDING READ FOR UPDATE EXISTS

Explanation: When a record is read from an INDEXED file which contains blocked records and which is open for DIRECT UPDATE, the record must be rewritten. Between the READ statement and the associated REWRITE statement, no other operation may be performed on the file.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE033I FILE name - NO COMPLETED READ EXISTS (INCORRECT NCP VALUE)

Explanation: This message may be issued because the correct NCP value has not been specified or it may be due to incorrect source code.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP

DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE034I FILE name - TOO MANY INCOMPLETE I/O OPERATIONS

Explanation: The number of incomplete I/O operations equals the NCP value.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE035I FILE name - EVENT VARIABLE ALREADY IN USE

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE036I FILE name - IMPLICIT OPEN FAILURE, CANNOT PROCEED

Explanation: There has been a failure in an implicit OPEN operation.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE037I FILE name - ATTEMPT TO REWRITE OUT OF SEQUENCE

Explanation: An intervening I/O statement occurs between a READ statement and a REWRITE statement referring to the same record.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE038I FILE name - ENDFILE FOUND

UNEXPECTEDLY IN MIDDLE OF DATA ITEM.

Explanation: The ERROR condition is raised when end-of-file is encountered before the delimiter when scanning list-directed or data-directed input, or if the field width in the format list of edit-directed input would take the scan beyond the end-of-file.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE039I FILE name - ATTEMPT TO CLOSE FILE NOT OPENED IN CURRENT TASK

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

I/O ON-conditions

All these conditions may be raised by the SIGNAL statement.

IHE100I FILE name - UNRECOGNIZABLE DATA NAME

Explanation:

Initiating ON-condition: NAME

1. GET DATA - name of data item found on external medium is not known at the time of the GET statement, or
2. GET DATA data list - name of data item found on external medium is not specified in the list.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE110I FILE name - RECORD CONDITION SIGNALED

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP

DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE111I FILE name - RECORD VARIABLE SMALLER THAN RECORD SIZE

Explanation: The variable specified in the READ statement INTO option allows fewer characters than exist in the record.

F format records:
a WRITE statement attempts to put a record smaller than the record size.

All formats:
a REWRITE attempts to replace a record with one of smaller size. (Note: This condition cannot be detected for U-format records read for UNBUFFERED or DIRECT files.)

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE112I FILE name - RECORD VARIABLE LARGER THAN RECORD SIZE

Explanation: The variable specified in the READ statement INTO option requires more characters than exist in the record; or a WRITE statement attempts to put out a record greater than the available record size; or a REWRITE statement attempts to replace a record with one of greater size.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE113I ATTEMPT TO WRITE/LOCATE ZERO LENGTH RECORD

Explanation: A WRITE or REWRITE statement attempts to put out a record of zero length, or a LOCATE statement attempts to get buffer space for a record of zero length; such records are used for end-of-file markers for direct access storage devices.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE114I FILE name - ZERO LENGTH RECORD READ

Explanation: A record of zero length has been read from a REGIONAL data set accessed in the DIRECT mode. This should not occur, unless the data set was created by another processor. A zero length record, on a direct access device, is an end-of-file signal.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE120I FILE name - PERMANENT INPUT ERROR

Explanation:

Initiating ON-CONDITION:
TRANSMIT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE121I FILE name - PERMANENT OUTPUT ERROR

Explanation:

Initiating ON-condition:
TRANSMIT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE122I FILE name - TRANSMIT CONDITION SIGNALED

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE130I FILE name - KEY CONDITION SIGNALED

Programmer Response: Probable user error. After making correction, recompile the program and execute the job

step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE131I FILE name - KEYED RECORD NOT FOUND

Explanation: READ, REWRITE, or DELETE statement specified record key which does not match with records of data set. If REGIONAL (2) or (3) data sets are employed, and the DD statement parameter LIMCT is used, then the record does not exist within the number of records or tracks searched, but may exist elsewhere.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE132I FILE name - ATTEMPT TO ADD DUPLICATE KEY

Explanation: WRITE statement specified a key value which already exists within data set.

1. INDEXED data sets: detected for both SEQUENTIAL and DIRECT access.
2. REGIONAL data sets: detected only for REGIONAL (1) and (2) SEQUENTIAL output.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE133I FILE name - KEY SEQUENCE ERROR

Explanation: WRITE statement specified, during creation of data set (OUTPUT SEQUENTIAL), a key which for:

1. INDEXED data sets is lower in binary collating sequence than prior key
2. REGIONAL data sets the relative record/track value is lower than that of prior key.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP

DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE134I FILE name - KEY CONVERSION ERROR

Explanation: WRITE, READ, REWRITE, or DELETE statement for REGIONAL data set specified character string key value whose relative record/track partition contains characters other than blank or the digits 0 through 9, or which contains only the character blank.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE135I FILE name - KEY SPECIFICATION ERROR

Explanation:

1. INDEXED: the KEYFROM or KEY expression may be the NULL string. Alternatively, RKP does not equal zero and the embedded key is not identical with that specified by the KEYFROM option (or the KEY option in the case of a rewrite statement). A third possibility is that an attempt has been made during SEQUENTIAL UPDATE to replace a record by one whose embedded key does not match that of the original record.
2. REGIONAL: as for INDEXED, or initial character of KEY or KEYFROM expression value is the value (8)'1'B.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE136I FILE name - KEYED RELATIVE RECORD/TRACK OUTSIDE DATA SET LIMIT

Explanation: WRITE, READ, REWRITE, or DELETE statement for REGIONAL data set specified a key whose relative record/track value exceeds the number of records or tracks assigned to the data set.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage

Editor/Loader Map and program listing available.

IHE137I FILE name - NO SPACE AVAILABLE TO ADD KEYED RECORD

Explanation: WRITE statement attempted to add record, but data set was full. If REGIONAL (2) or (3) data set, condition is raised if space within optional limits (DD parameter LIMCT) is unavailable.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE140I FILE name - END OF FILE ENCOUNTERED

Explanation:

Initiating ON-condition:
ENDFILE

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE150I FILE name - CANNOT BE OPENED, NO DD CARD

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE151I FILE name - CONFLICTING DECLARE AND OPEN ATTRIBUTES

Initiating ON-condition:
UNDEFINEDFILE

There is a conflict between the declared PL/I file attributes. For example:

Attribute	Conflicting Attributes
PRINT	INPUT, UPDATE, RECORD, DIRECT, SEQUENTIAL, TRANSIENT, BACKWARDS, BUFFERED, UNBUFFERED, EXCLUSIVE, KEYED
STREAM	UPDATE, RECORD, DIRECT, TRANSIENT, SEQUENTIAL, BACKWARDS, BUFFERED, UNBUFFERED, EXCLUSIVE, KEYED
EXCLUSIVE	INPUT, OUTPUT, SEQUENTIAL, TRANSIENT,

	BACKWARDS, BUFFERED, UNBUFFERED
DIRECT	SEQUENTIAL, TRANSIENT, BACKWARDS, BUFFERED, UNBUFFERED
UPDATE	INPUT, OUTPUT, BACKWARDS, TRANSIENT
OUTPUT	INPUT, BACKWARDS
BUFFERED	UNBUFFERED

Some attributes may have been supplied when a file is opened implicitly. Example of attributes implied by I/O statements are:

I/O Statement	Implied Attributes
DELETE	RECORD, DIRECT, UPDATE
GET	INPUT
LOCATE	RECORD, OUTPUT, SEQUENTIAL, BUFFERED
PUT	OUTPUT
READ	RECORD, INPUT
REWRITE	RECORD, UPDATE
UNLOCK	RECORD, DIRECT, UPDATE, EXCLUSIVE
WRITE	RECORD, OUTPUT

In turn, certain attributes may imply other attributes:

Attribute	Implied Attributes
BACKWARDS	RECORD, SEQUENTIAL, INPUT
BUFFERED	RECORD, SEQUENTIAL
DIRECT	RECORD, KEYED
EXCLUSIVE	RECORD, KEYED, DIRECT, UPDATE
KEYED	RECORD
PRINT	OUTPUT, STREAM
SEQUENTIAL	RECORD
UNBUFFERED	RECORD, SEQUENTIAL
UPDATE	RECORD

Finally, a group of alternate attributes has one of the group as a default. The default is implied if none of the group is specified explicitly or is implied by other attributes or by the opening I/O statement. The groups of alternates are:

Group	Default
STREAM|RECORD	STREAM
INPUT|OUTPUT|UPDATE	INPUT
SEQUENTIAL|DIRECT|TRANSIENT (RECORD files)	SEQUENTIAL
BUFFERED|UNBUFFERED (SEQUENTIAL files)	BUFFERED

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE152I FILE name - FILE TYPE NOT SUPPORTED

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

The user has attempted to associate a paper-tape device with a file that does not have the INPUT attribute.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE153I FILE name - BLOCKSIZE NOT SPECIFIED

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

Block size not specified on DD card, nor on environment. However, will never occur for PRINT file, because default block size is assumed.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE154I FILE name - UNDEFINEDFILE CONDITION SIGNALED

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE155I FILE name - ERROR INITIALIZING REGIONAL DATA SET

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

A REGIONAL data set, opened for DIRECT OUTPUT, cannot be properly formatted during the open process.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE156I FILE name - CONFLICTING ATTRIBUTE AND ENVIRONMENT PARAMETERS

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

Examples of conflicting parameters NAME block size is assumed. are:

ENVIRONMENT Parameter	File Attribute
No file organization parameter	KEYED
INDEXED|REGIONAL	STREAM
CONSECUTIVE	DIRECT| EXCLUSIVE
INDEXED	DIRECT OUTPUT
INDEXED|REGIONAL	OUTPUT without KEYED
Blocked records	UNBUFFERED
V-format records	BACKWARDS

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE157I FILE name - CONFLICTING ENVIRONMENT AND/OR DD PARAMETERS

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

One of the following conflicts exists:

1. F-format records have not been specified for a REGIONAL(1) or REGIONAL(2) file.
2. Blocked records have been specified with a REGIONAL file.
3. Track overflow was specified in the DD statement of a REGIONAL(3) file using U- or V-format records.
4. Track overflow was specified in the ENV attribute of a REGIONAL(3) file using U- or V-format records. The compile-time message was ignored.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE158I FILE name - KEYLENGTH NOT SPECIFIED

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

A keylength has not been specified for an INDEXED, REGIONAL(2), or REGIONAL(3) file that is being opened for OUTPUT.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls

IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE159I FILE name - INCORRECT BLOCKSIZE AND/OR LOGICAL RECORD SIZE IN STATEMENT NUMBER xxx

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

One of the following situations exists:

1. F-format records
   a. The specified block size is less than the logical record length.
   b. The specified block size is not a multiple of the logical record length.
2. V-format records
   a. The specified block size is less than the logical record length + 4.
   b. The logical record length is less than 14 for a RECORD file or 15 for a STREAM file.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE160I FILE name - LINESIZE GREATER THAN IMPLEMENTATION DEFINED MAXIMUM LENGTH

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

The implementation-defined maximum linesize is:

F-format records 32759

V-format records 32751

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE161I FILE name - CONFLICTING ATTRIBUTE AND DD PARAMETERS

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

The user has attempted to associate a file with the BACKWARDS attribute with a device that is not a magnetic tape device.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there

is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

Computational Errors

IHE200I rtname - X LE 0 IN SQRT(X)

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE202I rtname - X LE 0 IN LOG(X) OR LOG2(X) OR LOG10(X)

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE203I rtname - ABS(X) GE (2**50)*K IN SIN(X) OR COS(X) (K=PI) OR SIND(X) OR COSD(X) (K=180)

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE204I rtname - ABS(X) GE (2**50)*K IN TAN(X) (K=PI) OR TAND(X) (K=180)

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE206I rtname - X=Y=0 IN ATAN(Y,X) AND ATAND(Y,X)

Programmer Response: Probable user error. After making correction, recompile the program and execute the job

step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE208I rtname - ABS(X) GT 1 IN ATANH(X)

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE209I rtname - X=0, Y LE 0 IN X**Y

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE210I rtname - X=0, Y NOT POSITIVE REAL IN X**Y

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE211I rtname - Z=+I OR -I IN ATAN(Z) OR Z=+1 OR -1 IN ATANH(Z)

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE212I rtname - ABS(X) GE (2**18)*K IN SIN(X) OR COS(X) (K=PI) OR SIND(X) OR COSD(X) (K=180)

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE213I rtname - ABS(X) GE (2**18)*K IN TAN(X) (K=PI) OR TAND(X) (K=180)

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

List of Routine Names

IHESQS	Short float square root
IHELNS	Short float logarithm
IHETNS	Short float tangent
IHEATS	Short float arctan
IHESNS	Short float sine and cosine
IHEHTS	Short float hyperbolic arctan
IHESQL	Long float square root
IHELNL	Long float logarithm
IHETNL	Long float tangent
IHEATL	Long float arctan
IHESNL	Long float sine and cosine
IHEHTL	Long float hyperbolic arctan
IHEXIS	Short float integer exponentiation
IHEXIL	Long float integer exponentiation
IHEXXS	Short float general exponentiation
IHEXXL	Long float general exponentiation
IHEXIW	Short float complex integer exponentiation
IHEXIZ	Long float complex integer exponentiation
IHEXXW	Short float complex general exponentiation
IHEXXZ	Long float complex general exponentiation
IHEATW	Short float complex arctan and hyperbolic arctan
IHEATZ	Long float complex arctan and hyperbolic arctan

Computational ON-Conditions

All these conditions may be raised by the SIGNAL statement.

IHE300I OVERFLOW

Explanation: This condition is raised, by Library routines or by compiled code, when the

exponent of a floating-point number exceeds the permitted maximum, as defined by implementation.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE310I SIZE

Explanation: This condition is raised, by Library routines or by compiled code, when assignment is attempted where the number to be assigned will not fit into the target field. This condition can be raised by allowing the fixed overflow interrupt to occur on account of SIZE. If associated with I/O, then "FILE name" will be inserted between the message number and the text.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE320I FIXEDOVERFLOW

Explanation: This condition is raised, by Library routines or by compiled code, when the result of a fixed-point binary or decimal operation exceeds the maximum field width as defined by implementation.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE330I ZERODIVIDE

Explanation: This condition is raised, by Library routines or by compiled code, when an attempt is made to divide by zero, or when the quotient exceeds the precision allocated for the result of a division. The condition can be raised by hardware interrupt or by special coding.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP

DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE340I UNDERFLOW

Explanation: This condition is raised, by Library routines or by compiled code, when the exponent of a floating-point number is smaller than the implementation-defined mimimum. The condition does not occur when equal floating-point numbers are subtracted.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the UNDERFLOW condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE350I STRINGRANGE

Explanation: This condition is raised by library routines when an invalid reference by the SUBSTR built-in function or pseudo-variable has been detected.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVLE=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the STRINGRANGE condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE360I AREA CONDITION RAISED IN ALLOCATE STATEMENT

Explanation: There is not enough room in the area in which to allocate the based variable.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE361I AREA CONDITION RAISED IN ASSIGNMENT STATEMENT

Explanation: There is not enough room in the area to which the based variable is being assigned.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the

Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE362I AREA SIGNALED

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

Structure and Array Errors

IHE380I IHESTR - STRUCTURE OR ARRAY LENGTH GE 16**6 BYTES

Explanation: During the mapping of a structure or array, the length of the structure or array has been found to be greater than or equal to 16**6 bytes.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE381I IHESTR - VIRTUAL ORIGIN OF ARRAY GE 16**6 OR LE -16**6

Explanation: During the mapping of a structure, the address of the element with zero subscripts in an array, whether it exists or not, has been computed to be outside the range (-16**6 to +16**6).

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE382I IHESTR - UPPER BOUND LESS THAN LOWER BOUND

Explanation: During the mapping of an array or structure, an upper bound of a dimension has been found to be less than the corresponding lower bound. If only an upper bound was declared then it may currently be less than one, the implied lower bound.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was

specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

Control Program Restrictions

IHE400I DELAY STATEMENT EXECUTED - NO TIMER FUNCTION IN SYSTEM

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE401I TIME STATEMENT EXECUTED - NO TIMER FUNCTION IN SYSTEM

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

Condition Type ON-Conditions

IHE500I SUBSCRIPTRANGE

Explanation: This condition is raised, by library routines or by compiled code, when a subscript is evaluated and found to lie outside its specified bounds, or by the SIGNAL statement.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE501I CONDITION

Explanation: This condition is raised by execution of a SIGNAL (identifier) statement, referencing a programmer-specified EXTERNAL identifier.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls

IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

Errors Associated with Tasking

The following errors are associated with execution of the CALL, READ, or WRITE statement with TASK option; with the WAIT statement; with the use of TASK or EVENT variables; with the PRIORITY pseudo-variable or built-in function; or with the COMPLETION pseudo-variable.

IHE550I ATTEMPT TO WAIT ON AN INACTIVE AND INCOMPLETE EVENT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE551I TASK VARIABLE ALREADY ACTIVE

Explanation: Task variable is already associated with an active task.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE552I EVENT ALREADY BEING WAITED ON

Explanation: During the execution of a WAIT statement, in order to complete the required number of events, an event must not be waited on which is already being waited on in another task.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE553I WAIT ON MORE THAN 255 INCOMPLETE EVENTS

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE554I ACTIVE EVENT VARIABLE AS ARGUMENT TO COMPLETION PSEUDO-VARIABLE

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE555I INVALID TASK VARIABLE AS ARGUMENT TO PRIORITY PSEUDO-VARIABLE

Explanation: The task variable specified was active and not associated with the current task or one of its immediate subtasks.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE556I EVENT VARIABLE ACTIVE IN ASSIGNMENT STATEMENT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE557I EVENT VARIABLE ALREADY ACTIVE

Explanation: Event variable is already associated with an active task.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE558I ATTEMPT TO WAIT ON AN I/O EVENT IN WRONG TASK

Explanation: An I/O event can be waited on only in the same task as the statement which initiated the I/O operation with which the event is associated.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE559I UNABLE TO CALL TASK DUE TO INSUFFICIENT MAIN STORAGE.

Explanation: Insufficient main storage available for the additional task being called.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE560I UNABLE TO CALL TASK DUE TO MORE THAN 255 TASKS ACTIVE.

Explanation: The program attempted to call a subtask which would have made the number of active tasks greater than 255.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

In the following group of messages, hhh is a hexadecimal number.

IHE571I TASK (name) TERMINATED. COMPLETION CODE= hhh.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE572I TASK (name) TERMINATED. COMPLETION CODE = hhh. EVENT VARIABLE OVERWRITTEN OR DESTROYED.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE573I TASK (name) TERMINATED. COMPLETION CODE = hhh. TASK VARIABLE OVERWRITTEN OR DESTROYED.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE574I TASK (NAME) TERMINATED. COMPLETION CODE = hhh. INVALID FREE STATEMENT.

Explanation: The FREE statement freed, or tried to free, storage to which it is not applicable.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE575I TASK (name) TERMINATED. COMPLETION CODE = hhh. DISPLAY STATEMENT. REPLY NOT WAITED FOR.

Explanation: The task terminated normally without waiting for a reply from a DISPLAY statement with the REPLY option.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE576I TASK (name) TERMINATED. COMPLETION CODE = hhh. TOO MUCH MAIN STORAGE REQUESTED.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls

IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE577I TASK (name) TERMINATED WHILE STILL ACTIVE -- END OF BLOCK REACHED IN ATTACHING TASK.

Explanation: The attaching task reached:

1. An EXIT statement, or
2. The end of the block in which the subtask was attached

while the subtask was still active.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE579I TASK (name) TERMINATED. COMPLETION CODE = hhh. ABNORMAL TERMINATION DURING PUT STATEMENT.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

Conversion ON-Conditions

Conversion errors occur most often on input, either owing to an error in the input data, or because of an error in a format list. For example, in edit-directed input, if the field width of one of the items in the data list is incorrectly specified in the format list, the input stream will get out of step with the format list and a conversion error is likely to occur.

IHE600I CONVERSION CONDITION SIGNALED

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE601I CONVERSION ERROR IN F-FORMAT INPUT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job

statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE602I CONVERSION ERROR IN E-FORMAT INPUT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE603I CONVERSION ERROR IN B-FORMAT INPUT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE604I ERROR IN CONVERSION FROM CHARACTER STRING TO ARITHMETIC

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE605I ERROR IN CONVERSION FROM CHARACTER STRING TO BIT STRING

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE606I ERROR IN CONVERSION FROM CHARACTER STRING TO PICTURED CHARACTER STRING

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE607I CONVERSION ERROR IN P-FORMAT INPUT (DECIMAL)

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE608I CONVERSION ERROR IN P-FORMAT INPUT (CHARACTER)

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE609I CONVERSION ERROR IN P-FORMAT INPUT (STERLING)

Note: When condition was due to an I/O conversion, then "FILE name" will be inserted between the message number and the text. Also, when the I/O conversion error was due to a TRANSMIT error, the word (TRANSMIT) is inserted between the file name and the text.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

Conversion Errors, Non-ON-Type

IHE700I INCORRECT E(W,D,S) SPECIFICATION

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE701I F FORMAT W SPECIFICATION TOO SMALL

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE702I A FORMAT W UNSPECIFIED AND LIST ITEM NOT TYPE STRING

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE703I B FORMAT W UNSPECIFIED AND LIST ITEM NOT TYPE STRING

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE704I A FORMAT W UNSPECIFIED ON INPUT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE705I B FORMAT W UNSPECIFIED ON INPUT

Explanations: Messages 700 to 705 reveal that an EDIT operation was incorrectly specified.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there

is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE706I UNABLE TO ASSIGN TO PICTURED CHARACTER STRING

Explanation: A source datum which is not a character string cannot be assigned to a pictured character string because of a mismatch with the PIC description of the target.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE798I ONSOURCE OR ONCHAR PSEUDOVARIABLE USED OUT OF CONTEXT

Explanation: This message is printed and the ERROR condition raised if an ONSOURCE or ONCHAR pseudo-variable is used outside an ON-unit, or in an ON-unit other than either a CONVERSION ON-unit or an ERROR or FINISH ON-unit following from system action for CONVERSION.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE799I RETURN ATTEMPTED FROM CONVERSION ON-UNIT BUT SOURCE FIELD NOT MODIFIED

Explanation: A CONVERSION ON-unit has been entered as a result of an invalid conversion, and an attempt has been made to return, and hence reattempt the conversion, without using one or other of the pseudo-variables ONSOURCE or ONCHAR to change the invalid character.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

Non-Computational Program Interrupt Errors

Certain program interrupts may occur in a PL/I program because the source program has an error which is severe but which cannot be detected until execution time. An example is a call to an unknown procedure, which will result in an illegal

operation program interrupt. Other program interrupts, such as addressing, specification, protection, and data interrupts, may arise if PL/I control blocks have been destroyed. This can occur if an assignment is made to an array element whose subscript is out of range, since, if SUBSCRIPTRANGE has not been enabled, the compiler does not check array subscripts; a program interrupt may occur at the time of the assignment or at a later stage in the program. Similarly, an attempt to use the value of an array element whose subscript is out of range may cause an interrupt.

Care must be taken when parameters are passed to a procedure. If the data attributes of the arguments of the calling statement do not agree with those of the invoked entry point, or if an argument is not passed at all, a program interrupt may occur.

The use of the value of a variable that has not been initialized, or has had no assignment made to it, or the use of CONTROLLED variables that have not been allocated, may also cause one of these interrupts.

IHE800I INVALID OPERATION

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE801I PRIVILEGED OPERATION

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE802I EXECUTE INSTRUCTION EXECUTED

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE803I PROTECTION VIOLATION

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage

Editor/Loader Map and program listing available.

IHE804I ADDRESSING INTERRUPT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE805I SPECIFICATION INTERRUPT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE806I DATA INTERRUPT

Explanation: This condition can be caused by an attempt to use the value of a FIXED DECIMAL variable when no prior assignment to, or initialization of, the variable has been performed.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

Model 91 and Model 195 Object-Time Diagnostic Messages

After a multiple-exception imprecise interrupt on a Model 91 or Model 195 certain exceptions will remain unprocessed if the ERROR condition is raised before all the exceptions have been handled. If the program subsequently is terminated as a direct result of the ERROR condition being raised in these circumstances, one or more of the following messages will be printed out.

IHE810I PROTECTION EXCEPTION UNPROCESSED AFTER MULTIPLE-EXCEPTION IMPRECISE INTERRUPT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE811I ADDRESSING EXCEPTION UNPROCESSED AFTER MULTIPLE-EXCEPTION IMPRECISE INTERRUPT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE812I SPECIFICATION EXCEPTION UNPROCESSED AFTER MULTIPLE-EXCEPTION IMPRECISE INTERRUPT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE813I DATA EXCEPTION UNPROCESSED AFTER MULTIPLE-EXCEPTION IMPRECISE INTERRUPT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE814I ZERODIVIDE UNPROCESSED AFTER MULTIPLE-EXCEPTION IMPRECISE INTERRUPT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.
- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE815I OVERFLOW UNPROCESSED AFTER MULTIPLE-EXCEPTION IMPRECISE INTERRUPT

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the

Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

Storage Management Errors

The following errors are associated with the handling of storage and transfer of control out of blocks. In some cases, these errors are a result of program error, but it is possible that the messages may be printed because the save area chain, allocation chain, or pseudo-register vector have been overwritten.

IHE900I TOO MANY ACTIVE ON-UNITS AND ENTRY PARAMETER PROCEDURES

Explanation: There is an implementation limit to the number of ON-units and/or entry parameter procedures which can be active at any time. An entry parameter procedure is one that passes an entry name as parameter to a procedure it calls. The total permissible number of these ON-units/entry parameter procedures is 127.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

IHE902I GOTO STATEMENT REFERENCES A LABEL IN AN INACTIVE BLOCK

Explanation: The label referred to cannot be found in any of the blocks currently active in the current task; blocks are not freed. The statement number and offset indicate the GO TO statement causing the error.

Programmer Response: Probable user error. After making correction, recompile the program and execute the job step again. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the job statement, and that there is an active on-unit in the Main procedure for the ERROR condition which calls IHEDUMP and that a PL1DUMP DD statement was included for the failing job step.

- Have the associated jobstream, Linkage Editor/Loader Map and program listing available.

Index

READER'S COMMENT FORM

IBM System/360
PL/I (F) Programmer's Guide

Order No. GC28-6594-7

- How did you use this publication?

  As a reference source ..... ☐
  As a classroom text ..... ☐
  As a self-study text ..... ☐

- Based on your own experience, rate this publication . . .

	Very Good	Good	Fair	Poor	Very Poor
As a reference source:	.....	.....	.....	.....	.....
As a text:	.....	.....	.....	.....	.....

- What is your occupation? .....

- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

- Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A.

YOUR COMMENTS PLEASE . . . .

This SRL manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note : Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Cut Along Line

fold fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N.Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY...

IBM Corporation
112 East Post Road
White Plains, N.Y. 10601

Attention: Department 813 (HP)

fold fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

IBM Systems/360 Printed in U.S.A. GC28-6594-7

IBM System/360 Printed in U.S.A. GC28-6594-7

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]