

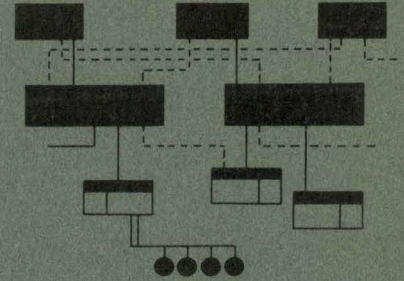
GE-625/635

● COBOL



Information Systems
Equipment

497-2727 402



GENERAL  ELECTRIC

GE-625/635 COBOL REFERENCE MANUAL

September 1964
Rev. April 1969

INFORMATION SYSTEMS

GENERAL  **ELECTRIC**

PREFACE

This reference manual was prepared to allow GE-625 and GE-635 users, with programmers experienced in COBOL programming principles, to program in the COBOL System environment. It is written as a reference manual and contains general information on the concepts for the organization of a COBOL program, the elements of the language, the rules for combining these elements into meaningful instructions, and the methods for preparing data for use in a COBOL program. In addition, this manual specifies in detail the exact forms which the programmer must follow in writing instructions in the COBOL language for the GE-625 and the GE-635.

A compiler is being prepared to accept the COBOL-61 Extended source programs described in this manual. The Department of Defense specifications for COBOL-61 Extended are closely followed in COBOL for the GE-625 and the GE-635. The portions of COBOL-61 Extended not included in this manual are not available at this time.

This edition of the COBOL Reference Manual, dated April 1969, supersedes the previous edition, dated January, 1967, and includes the following Technical Information Bulletins (TIB's) issued for the previous edition, CPB-1007E: TIB 600-185, TIB 600-216, TIB 600-235, and TIB 600-236.

Suggestions and criticisms relative to form, content, purpose, or use of this manual are invited. Comments may be sent on the Document Review Sheet in the back of this manual or may be addressed directly to General Electric Company, Information Systems Equipment Division, C-77, 13430 North Black Canyon Highway, Phoenix, Arizona, 85029.

© 1964, 1965, 1967, 1969 by General Electric Company

(2M 8-69)

TABLE OF CONTENTS

	Page
I. ACKNOWLEDGEMENT	I-1
II. GENERAL DESCRIPTION OF "COBOL"	II-1
III. REFERENCE FORMAT	
GENERAL DESCRIPTION	III-1
SEQUENCE NUMBERS	III-1
IDENTIFICATION DIVISION	III-1
ENVIRONMENT DIVISION	III-2
DATA DIVISION	III-3
PROCEDURE DIVISION	III-4
IV. NOTATION USED IN VERB AND ENTRY FORMATS IN THIS REPORT	
BRACES	IV-1
BRACKETS	IV-1
REQUIRED WORDS	IV-1
OPTIONAL WORDS	IV-1
LOWER CASE WORDS	IV-1
CONNECTIVES	IV-1
PERIOD	IV-1
V. CHARACTERS AND WORDS	
COMPLETE CHARACTER SET	V-1
CHARACTERS USED FOR WORDS	V-1
DEFINITION OF WORDS	V-1
TYPES OF WORDS	V-2
Nouns	V-2
Data-Names	V-2
Condition-Names	V-2
Procedure-Names	V-3
Literals	V-3
Numeric Literal	V-3
Figurative Constants	V-4
Special Register	V-6
Verbs	V-6
Reserved Words	V-6
Qualifiers	V-7
Subscripts	V-8
DISTINCTION BETWEEN SUBSCRIPTS AND QUALIFIERS	V-10
Series Connectives	V-10
COMPLETE LIST OF RESERVED WORDS	V-11

	Page
VI. DATA DIVISION	
GENERAL DESCRIPTION	VI-1
ORGANIZATION	VI-1
FILE DESCRIPTION ENTRY	VI-2
ENTRY FORMATS	VI-3
General Notes	VI-3
Specific Formats	VI-3
File Description Complete Entry	VI-4
BLOCK Size	VI-6
COPY	VI-7
DATA RECORDS	VI-8
FILE Size	VI-9
LABEL RECORDS	VI-10
RECORD Size	VI-11
RECORDING MODE	VI-12
REPORT(S)	VI-13
SEQUENCED	VI-14
VALUE	VI-15
RECORD DESCRIPTION	VI-16
Concept of Levels	VI-17
Concept of a Computer Independent Detailed Data Description	VI-19
Derivation of External and Internal Formats	VI-19
Algebraic Signs	VI-19
Item Alignment and Spacing on Fixed Word Length Computers	VI-19
Report Editing	VI-20
ENTRY FORMATS	VI-20
General Notes	VI-20
Specific Formats	VI-20
RECORD DESCRIPTION	VI-21
Complete Entry Skeleton	VI-21
CLASS	VI-22
COPY	VI-24
Data-name	VI-25
Editing Clauses	VI-26
JUSTIFIED	VI-28
Level Number	VI-29
OCCURS	VI-30
PICTURE	VI-31
POINT LOCATION	VI-38
RANGE	VI-39
REDEFINES	VI-40
RENAMES	VI-41
SIGN	VI-42
SIZE	VI-43
SYNCHRONIZED	VI-44
USAGE	VI-45

	Page
VALUE	VI-47
Specific Entry for a Condition-name	VI-48
REPORT WRITER	VI-50
GENERAL DESCRIPTION	VI-50
Report Section	VI-50
Report Name Description Entry	VI-50
Report Group Description Entry	VI-51
DEFINITIONS	VI-51
LINE COUNTER	VI-53
PAGE COUNTER	VI-54
ENTRY FORMATS	VI-54
General Notes	VI-54
Specific Formats	VI-54
RD Entry	VI-55
CODE	VI-56
CONTROL(S)	VI-57
COPY	VI-58
PAGE LIMIT(S)	VI-59
Report Group Entries	VI-61
CLASS	VI-63
COLUMN NUMBER	VI-64
COPY	VI-65
DATA-NAME	VI-66
EDITING CLAUSES	VI-67
GROUP INDICATE	VI-68
JUSTIFIED	VI-69
LEVEL NUMBER	VI-70
LINE NUMBER	VI-71
NEXT GROUP	VI-72
PICTURE	VI-73
POINT LOCATION	VI-74
SIZE	VI-74
SOURCE-SUM-VALUE	VI-75
SOURCE	VI-75
SUM	VI-75
VALUE	VI-76
TYPE	VI-77
USAGE	VI-82
SUMMARY	VI-83
FILE SECTION	VI-83
Specification and Handling of Labels	VI-83
WORKING STORAGE SECTION	VI-86
Organization	VI-86
Non-Contiguous Working Storage	VI-86
Working Storage Records	VI-87
Initial Values	VI-87
Condition-Names	VI-87

	Page
CONSTANT SECTION	VI-88
Organization	VI-88
Non-Contiguous Constant Section	VI-88
Constant Records	VI-89
VALUE of Constants	VI-89
Condition-Names	VI-89
Table of Constants	VI-89
VII. PROCEDURE DIVISION	
GENERAL DESCRIPTION	VII-1
RULES OF PROCEDURE FORMATION	VII-1
STATEMENTS	VII-1
Imperative Statements	VII-1
Conditional Statements	VII-1
Compiler Directing Statements	VII-2
SENTENCES	VII-2
Imperative Sentences	VII-2
Conditional Sentences	VII-2
Compiler Directing Sentences	VII-2
SENTENCE PUNCTUATION	VII-3
Verb Formats	VII-3
Sentence Formats	VII-3
SENTENCE EXECUTION	VII-3
Imperative Sentences	VII-4
Conditional Sentences	VII-4
Compiler Directing Sentences	VII-4
CONTROL RELATIONSHIP BETWEEN PROCEDURES	VII-4
CONDITIONALS	VII-5
GENERAL DESCRIPTION	VII-5
CONDITIONS	VII-5
Simple Conditions	VII-5
Standard Collating Sequence	VII-7
Commercial Collating Sequence	VII-8
Compound Conditions	VII-10
Abbreviations	VII-12
FORMING COMPOUND CONDITIONS	VII-17
DECLARATIVES	VII-17
COMPILER DIRECTING DECLARATIVES	VII-18
FORMULAS	VII-18
Basic Operators	VII-18
VERBS	VII-20
SPECIFIC VERB FORMATS	VII-20
ACCEPT	VII-21
ADD	VII-22
ALTER	VII-24

	Page
CALL	VII-24.1
CLOSE	VII-25
COMPUTE	VII-28
COPY	VII-28.1
DISPLAY	VII-29
DIVIDE	VII-30
ENTER	VII-31
EXAMINE	VII-38
EXIT	VII-40
GENERATE	VII-41
GO	VII-43
INITIATE	VII-44
MOVE	VII-45
MULTIPLY	VII-48
NOTE	VII-49
OPEN	VII-50
PERFORM	VII-51
READ	VII-58
RELEASE	VII-60
RETURN	VII-61
SORT	VII-62
STOP	VII-65
SUBTRACT	VII-66
TERMINATE	VII-67
USE	VII-68
WRITE	VII-70

VIII. ENVIRONMENT DIVISION

GENERAL DESCRIPTION	VIII-1
STRUCTURE	VIII-1
CONFIGURATION SECTION	VIII-2
SOURCE-COMPUTER	VIII-2
OBJECT-COMPUTER	VIII-3
SPECIAL-NAMES	VIII-4
INPUT-OUTPUT SECTION	VIII-5
FILE CONTROL	VIII-5
I-O-CONTROL	VIII-7

IX. IDENTIFICATION DIVISION

GENERAL DESCRIPTION	IX-1
ORGANIZATION	IX-1
PROGRAM-ID	IX-2
DATE-COMPILED	IX-3

X. SOURCE-PROGRAM SEGMENTATION

INTRODUCTION	X-1
SEGMENTS	X-2
SECTIONS	X-2
DATA FILE COMMUNICATIONS	X-2
WORKING-STORAGE COMMUNICATIONS	X-3
CONSTANT COMMUNICATION	X-4
PROCEDURAL COMMUNICATIONS	X-4
DATA COMPATIBILITY	X-6
EXAMPLE OF MULTI-SEGMENT JOB	X-7

APPENDIXES

A. COMPUTATIONAL ITEM FORMATS	A-1
B. COBOL FILE FORMATS	B-1
C. PROCESSING NONLABELED MULTIPLE REEL FILES	C-1
D. PROCESSING STRANGER FILES VIA COBOL	D-1
E. COBOL EFFICIENCY TECHNIQUES	E-1
F. COBOL REPORT WRITER OBJECT MODEL	F-1
G. COBOL COMPILATIONS	G-1
H. SQUEEZE CODING IN COBOL SORTS	H-1
I. OPTIONAL COMPILATION OF STATEMENTS IN GE-625/635 COBOL SOURCE PROGRAMS	I-1
J. USE OF THE COBOL SORT FEATURE	J-1
K. OPTIMIZING THE COMPILATION PROCESS FOR GE-625/635 COBOL SOURCE PROGRAMS	K-1
L. USE OF WRITE VERB FOR LISTING FILES	L-1
M. REPORT WRITER PRE AND POST-SLEW CALCULATION	M-1
N. REPORT WRITER TABLE CAPACITY	N-1
O. COBOL EXAMPLES	O-1
P. COBOL SOURCE PROGRAM ORDER	P-1
Q. USE OF A LIBRARY FOR COPY	Q-1

INDEX

I. ACKNOWLEDGEMENT

This publication is based on the COBOL System developed in 1959 by a committee composed of government users and computer manufacturers. The organizations participating in the original development were:

Air Materiel Command, United States Air Force
Bureau of Standards, Department of Commerce
David Taylor Model Basin, Bureau of Ships, U. S. Navy
Electronic Data Processing Division, Minneapolis-Honeywell Regulator Co.
Burroughs Corporation
International Business Machines Corporation
Radio Corporation of America
Sylvania Electric Products, Inc.
Univac Division of Sperry-Rand Corporation

In addition to the organizations listed above, the following other organizations participated in the work of the Maintenance Group:

Allstate Insurance Company
Bendix Corporation, Computer Division
Control Data Corporation
DuPont Company
General Electric Company
General Motors Corporation
Lockheed Aircraft Corporation
National Cash Register Company
Philco Corporation
Royal McBee Corporation
Standard Oil Company (N.J.)
United States Steel Corporation

This manual is the result of contributions made by all of the above mentioned organizations. No warranty, expressed or implied, is made by any contributor or by the committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

It is reasonable to assume that a number of improvements and additions will be made to COBOL. Every effort will be made to insure that the improvements and corrections will be made in an orderly fashion, with due recognition of existing users' investments in programming. However, this protection can be positively assured only by individual implementors.

Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures and the methods for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

The authors and copyright holders of the copyrighted material used herein: FLOW-MATIC (Trade-Mark of Sperry-Rand Corporation), Programming for the UNIVAC ® I and II, Data Automation System © 1958, 1959, Sperry-Rand Corporation; IBM Commercial Translator, Form No. F28-8013, copyrighted 1959 by IBM, FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell, have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals and similar publications.

Any organization interested in reproducing the COBOL report and initial specifications in whole or in part, using ideas taken from this report or utilizing this report as the basis for an instruction manual or any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention COBOL in acknowledgment of the source, but need not quote this entire section.

II. GENERAL DESCRIPTION OF COBOL

COBOL is the preferred language for use within Department of Defense agencies and associated contractors for the programming of business data processing applications. The COBOL language was developed by a group of computer users and manufacturers and first documentation distributed in April, 1960. Since its initial distribution, it has undergone a number of changes instigated by manufacturer experience with COBOL implementation and user experience with COBOL programming for computers of all sizes and configurations. These improvements are embodied in the current version of the language termed COBOL-61 Extended.

From a computer user's standpoint, standard computer compiler languages:

1. provide a quick means of program implementation;
2. reduce costs of converting from the computer of one manufacturer to those of another;
3. reduce time and effort required for retraining of programmers;
4. guarantee at least some measure of standard documentation in the case that format documentation is not immediately forthcoming;
5. provide an effective way of taking immediate advantage of hardware and compiler improvements.

The COBOL System is composed of two elements - the source program written in COBOL, and the compiler which translates this source program into an object program capable of running on a computer. This report, in general, considers only the source program and does not consider the second element (the compiler) directly. However, the specifications of a language obviously determine, to a large extent, the boundaries of a compiler. Therefore, the compiler is mentioned in certain cases to facilitate the explanation of the language.

A source program is used to specify the solution of a business data processing problem. The four elements of this specification are:

1. The identification of the program.
2. The description of the equipment being used in the processing.
3. The description of the data being processed.
4. The set of procedures which determine how the data is to be processed.

The COBOL System has a separate division within the source program for each of these elements. The names of these divisions are:

IDENTIFICATION
ENVIRONMENT
DATA
PROCEDURE

The purpose of the IDENTIFICATION DIVISION is to identify the Source Program and outputs of a compilation. In addition, the user may include the date that the program was written, the date that the compilation was accomplished and any other information which is desired.

The ENVIRONMENT DIVISION is that part of the source program which specifies the equipment being used. It contains descriptions of the computers to be used both for compiling the source program and for running the object program. Memory size, number of tape units, etc., are among many items that may be mentioned for a particular computer. Those aspects of a file which relate directly to hardware are described here. Because this division deals entirely with the specifications of the equipment being used, it is largely computer dependent.

The DATA DIVISION uses file and record descriptions to describe the files of data that the object program is to manipulate or create, and the individual logical records which comprise these files. The characteristics or properties of the data are described in relation to a Standard Data Format rather than an equipment oriented format. Therefore, this division is to a large extent computer-independent. While compatibility among computers cannot, in general, be absolutely assured, careful planning in the data layout will permit the same data descriptions, with minor modification, to apply to more than one computer.

The PROCEDURE DIVISION specifies the steps that the user wishes the computer to follow. These steps are expressed in terms of meaningful English words, statements, sentences, and paragraphs. This aspect of the overall system is often referred to as the "program"; in reality, it is only part of the total specification of the problem solution (that is, the program), and is insufficient, by itself, to describe the entire problem. This is true because repeated references must be made - either explicitly or implicitly - to information appearing in the other divisions. This division, more than any other, allows the user to express his thoughts in meaningful English. Concepts of verbs to denote actions, and sentences to describe procedures, are basic, as is the use of conditional statements to provide alternative paths of action.

The PROCEDURE DIVISION is essentially computer independent. The amount of inter-computer compatibility throughout the COBOL System varies with the division, and the users' effort expended to obtain this goal. In the PROCEDURE DIVISION, virtually no effort is needed to maintain compatibility among computers. In the DATA DIVISION, some care must be taken to minimize the loss of object program efficiency. In the ENVIRONMENT DIVISION, almost all information is computer-dependent and therefore, the compatibility is based on ease of understanding rather than direct transference. The IDENTIFICATION DIVISION, like the PROCEDURE DIVISION, should require virtually no effort to maintain compatibility.

III. REFERENCE FORMAT

GENERAL DESCRIPTION

The purpose of the Reference Format is to provide a standard way of writing COBOL programs. Throughout this chapter, definitions of the proper Reference Format of each part of a COBOL source program are given in terms of lines and character positions (or columns) on an input/output medium. The standard method of representing sentences, paragraphs, sections, and divisions will be shown. The basic principle behind the particular formats chosen is to allow the maximum amount of flexibility for individual tastes while still using one basic form.

There are four parts to the Reference Format, the IDENTIFICATION DIVISION, the ENVIRONMENT DIVISION, the DATA DIVISION, and the PROCEDURE DIVISION. These divisions must appear in the Reference Format in the order specified above.

SEQUENCE NUMBERS

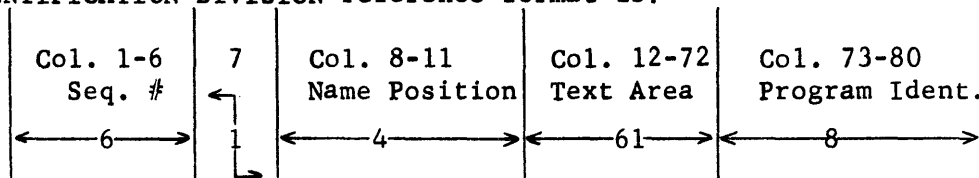
In order to facilitate corrections and changes, sequence numbers may be used. A sequence number consists of six (6) digits. It corresponds to a line on a coding form or listing.

IDENTIFICATION DIVISION

The IDENTIFICATION DIVISION provides a means to identify or label a COBOL source program. The only information required in this division is the PROGRAM-ID paragraph. Other information follows a standard format, but its inclusion is optional. Thus, the division may be composed of from one to seven paragraphs. The PROGRAM-ID paragraph must always appear as the first paragraph. Thereafter, any or all of the following fixed-name paragraphs may appear.

AUTHOR
DATE-WRITTEN
SECURITY
INSTALLATION
DATE-COMPILED
REMARKS

The IDENTIFICATION DIVISION reference format is:



The name of the division, and the names of the paragraphs within it, start in Column 8. The first line of this division contains its name, followed by a period, i.e.,

IDENTIFICATION DIVISION.

The text of each paragraph may start anywhere on the same line as the paragraph name, or on the next line, starting in Column 12. Any paragraph which occupies more than one line may be continued by starting in Column 12 on the next line. Any Program Identification may be placed in Column 73 through 80.

Any word or numeric literal can be split over two lines by placing a hyphen in Column 7 on the second line. In this case, any number of spaces may be left at the end of the first line following the first part of the split word or numeric literal, as they are ignored. Any non-numeric literal can also be split over two lines by placing a hyphen in Column 7 of the second line; but, in this case, any spaces appearing at the end of the first line are considered to be part of the literal. The continuation of the non-numeric literal on the second line must be immediately preceded by a quotation mark. The quotation mark can be any place at or to the right of the left-most allowable column (which is Column 12 except in indented lines in the DATA DIVISION). Any spaces which precede the continuation quotation mark are not considered part of the literal. The character * may be used in Column 7 to identify a line of comments. This allows remarks to be inserted at convenient places anywhere in the program. Such formats are recognized in all divisions of the program; however it is not recommended to place them between a sentence or entry which extends over several lines. If the * in Column 7 is immediately followed by the word EJECT in Columns 8-12, a special interpretation is made when the lines are listed, i.e. to slew a page after this line is encountered. If EJECT is not present in Columns 8-12, then the contents of Columns 8-72 have no effect except to be printed on the Reference Listing. A sample format for this division is as follows:

```

1      678
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. PAYROL .
000300 AUTHOR. JOHN DOE.
000600 DATE-WRITTEN. SEPTEMBER 5 1968
001100 REMARKS.
001200*
001300      INPUT FROM RUN 4 AND COLLATION WITH RUN 2 OUT
001400-      PUT TO RUN 6. THIS PROGRAM PROCESSES SALARI
001500-      ED EMPLOYEES ONLY.
001600*EJECT

```

ENVIRONMENT DIVISION

The reference format for the ENVIRONMENT DIVISION is the same as that of the IDENTIFICATION DIVISION. In addition to fixed paragraph names, there are also fixed section names. The name of the sections and paragraphs within it start

in Column 8. The first line of the division consists of its name, followed by a period, i.e.,

ENVIRONMENT DIVISION.

Each section name, like the division name appears as a single entry on a line by itself.

The rules for continuation of sentences and the splitting of words or literals over two lines are the same as those in the IDENTIFICATION DIVISION.

The SPECIAL-NAMES and the FILE-CONTROL paragraphs are each composed of several sentences, whereas the other paragraphs are each composed of one sentence only. The following is an example of a possible ENVIRONMENT DIVISION format:

```

1      678
000100 ENVIRONMENT DIVISION.
000200 CONFIGURATION SECTION.
000300 SOURCE-COMPUTER.           Computer-name...
000400 OBJECT-COMPUTER.         Computer-name...
001100 SPECIAL-NAMES.           Hardware-name...
001200 INPUT-OUTPUT SECTION.
001300 FILE-CONTROL.  SELECT file-name-1 ...
001400      SELECT file-name-2 ... SELECT file-name-3 ...
001500 I-O-CONTROL.             SAME ...
001600 *EJECT

```

DATA DIVISION

The basic unit in the DATA DIVISION is an entry. Each entry begins with a level number, followed by the name of a data item and a sequence of independent clauses describing the item. Each clause (except the last) may be terminated by a semicolon, followed by a space.

The user may choose whether to left-justify successive entries or indent them according to level number. If he chooses to indent all successive levels according to level number, he is restricted by the physical medium to a limited number of levels, but he obtains the advantage of displaying graphically the hierarchical structure of the data. The user may indent some levels and not others. The entries in the Reference Listing will be indented only if the input was indented. Under no circumstances does indentation affect the magnitude of a level number.

In the following examples, the six digit sequence number is represented by xxxxxx, and NN represents a two character level number. In a source program, single digit level numbers (e.g., 1 through 9) are written either as a space followed by a digit or as a zero followed by a digit.

The format for the DATA DIVISION, using indentation, is:

```
1      6 8 12                                     72
|      |
| xxxxxx NN data-name-1 ... .. |
| xxxxxx      NN data-name-2 ... .. |
| xxxxxx          NN data-name-3 ... .. |
```

The format for the DATA DIVISION, using no indentation, is:

```
| xxxxxx NN data-name-1 ... .. |
| xxxxxx NN data-name-2 ... .. |
| xxxxxx NN data-name-3 ... .. |
```

The first line, regardless of which form is used, consists of,

DATA DIVISION.

The sequence number appears at the left as in the format for the IDENTIFICATION DIVISION. The first level number starts in Column 8. If a single entry requires more than one line, the left margin for each line is the same, namely, the position under the first letter of the data-name associated with that entry. The rules for the splitting of words or literals over two lines and the use of * in Column 7 are the same as those in the IDENTIFICATION DIVISION. Column 7 may also be used to control optional compilation of entries by the debug level indicator (digits 0-9). A SPECIAL-NAMES sentence controls the effect of this indicator (see VIII-4). The absence of such control implies that the line has no effect except for printing on the Reference Listing. When the debug use is made of Column 7 for a line, no continuation by hyphen is possible on the next line.

When level numbers are to be indented, the recommended indentation is four (4) spaces to the right of the starting position of the previous level number. One or more spaces must be left between the level number and the following word. The word following the level number must begin at or to the right of Column 12.

An example of the DATA DIVISION Reference Format is on the next page.

```

1    678
000010 DATA DIVISION.
000020 FILE SECTION.
000100 FD MASTER-PAYROLL; LABEL RECORDS ARE
000200     STANDARD; DATA RECORDS ARE MASTER-
000300-    PAY; SEQUENCED ON BADGE-NUMBER.
000400     01 MASTER-PAY; SIZE IS 180.
000600         02 BADGE-NUMBER; SIZE IS 12 CH
000700-         ARACTERS; PICTURE IS
000800             AAAXXX999999.
000900         02 DATE; SIZE IS 6 CHARACTERS;
001000             CLASS IS NUMERIC.
001100         03 MONTH; SIZE IS 2 DIGITS.
001200         03 DAY; SIZE IS 2 CHARACTERS.
001300         03 YEAR; SIZE IS 2 CHARACTERS.
001400         02 GROSS-PAY; SIZE IS 6 CHARAC
001500-         TERS; PICTURE IS 0999V99.
001600*EJECT

```

PROCEDURE DIVISION

The reference format for the PROCEDURE DIVISION is the same as that of the IDENTIFICATION DIVISION.

The first line of the division consists of its name followed by a period, starting in Column 8, as follows:

```
PROCEDURE DIVISION.
```

If a section has been designated, the section-name starts in Column 8, followed by a space, the word SECTION, a period and a space.

A paragraph consists of one or more successive sentences, the first of which must be preceded by a paragraph-name. The name starts in Column 8, and is followed immediately by a period and a space. The first sentence of the paragraph may begin anywhere on the same line as the paragraph-name, or in Column 12 on the next line. A new paragraph is determined by the appearance of another paragraph-name. Note that a paragraph may consist of only a single sentence in some cases. With the exception of DECLARATIVE sentences, every sentence must be part of a paragraph.

Any sentence which occupies more than one line may be continued by starting in Column 12 on the next line. If a word or literal must be split over two lines, this will be indicated by placing a hyphen in Column 7 of the second line. Other rules applicable to the splitting of words and literals are the same as those for the IDENTIFICATION DIVISION. If the user prefers not to split a word or literal, he may start the word or literal on the next line. The use of * in Column 7 for comments or *EJECT for page slewing are the same as those for the IDENTIFICATION DIVISION.

Additionally, Column 7 may contain any of the digits 0-9 (debug indicator) which control optional compilation of lines by means of a sentence in the SPECIAL-NAMES paragraph (see VIII-4). Unless such a SPECIAL-NAMES entry is present, all such lines are ignored except for printing on the Reference Listing. However, when this use is made of Column 7 on a line, the use of Column 7 for continuation is not available.

When present, Declaratives form the first part of the PROCEDURE DIVISION, as stated in Chapter VII. The group of Declaratives must be preceded by the key word DECLARATIVES and a period, and must be followed by the key words, END DECLARATIVES and a period. These key words must begin in Column 8 of the Reference Format.

IV. NOTATION USED IN VERB AND ENTRY FORMATS IN THIS REPORT

BRACES

When words or phrases are enclosed in braces { }, a choice of one of the entries must be made.

BRACKETS

Words or phrases enclosed in square brackets [] represent options. The entry may be included in the statement if the option is desired; it should otherwise be omitted.

REQUIRED WORDS

Upper case words which are underlined are required whenever the statement or phrase in which they appear is used; they must appear in the source-program exactly as shown in the format illustration, except of course for the underline.

OPTIONAL WORDS

Upper case words which are not underlined are optional, and may be included in the source-program for the sake of readability, or else omitted. If they are used, they must appear in the source-program exactly as shown in the format illustration. They do not affect the result of the compilation.

LOWER CASE WORDS

Lower case words are generic terms, indicating the type of word (such as a data-name or a procedure-name or a literal) which must be supplied in that format position by the programmer.

CONNECTIVES

When two or more phrases or lower case words are written in a series, commas are shown as connectives. Wherever a comma is shown in the formats, except in a subscript, it may be omitted or replaced by either "AND", ", AND", "OR", ", OR". A comma must always be followed by at least one space (unless it is the last character of a line). Successive words on a line must be separated by at least one space, but as many spaces as desired may be used for this purpose.

PERIOD

When a period is shown in a format, it must appear in the same position whenever the statement is used in the source-program. A period must always be followed by at least one space (unless it is the last character of a line).

V. CHARACTERS AND WORDS

COMPLETE CHARACTER SET

The complete COBOL character set consists of the following 51 characters:

0, 1, ..., 9
 A, B, ..., Z
 blank or space
 + plus sign
 - minus sign or hyphen
 * asterisk
 / stroke (virgule or slash)
 = equal sign
 \$ dollar sign
 , comma
 . period or decimal point
 ; semicolon
 " quotation mark
 (left parenthesis
) right parenthesis
 > "greater than" symbol
 < "less than" symbol

CHARACTERS USED FOR WORDS

The character set for words consists of the following 37 characters:

0, 1, 9
 A, B, Z
 - (hyphen or minus)

Note particularly, that "blank" or "space" is not an allowable character for a word, but is used to separate words. Where a "blank" or "space" is employed, more than one may be used, except for the restrictions in the Reference Format. Groups of characters selected from the 37 characters are called "words".

DEFINITION OF WORDS

A word is composed of a combination of not more than 30 characters chosen from the following set of 37 characters:

0 - 9
 A - Z
 - (hyphen)

A word is ended by a space, or by either a period, right parenthesis, comma, or semicolon, followed by a space. A word may not begin or end with a hyphen.

The use of punctuation characters in connection with words is as follows:

A period, comma, and semicolon, when used, must always immediately follow a word, but they, in turn, must be followed by a space. A left parenthesis must not be followed immediately by a space, and a right parenthesis must not be immediately preceded by a space.

A beginning quotation mark must not be followed by a space, or an ending quotation mark preceded by a space, unless the spaces are desired in the literal.

TYPES OF WORDS

Nouns

A noun is a single word which is one of the following:

- data-name
- condition-name
- procedure-name
- literal
- figurative constant
- special register name
- special names (mnemonic-names)

A noun may contain hyphens for readability purposes. For example:

- quantity-on-hand
- stock-number

are legitimate nouns, whereas,

- stocknumber-

is not, since a word must not end with a hyphen.

DATA-NAMES

A data-name is a word containing at most 30 characters and with at least one alphabetic character. It identifies data specified in the data description, or the area which contains the data referenced.

Condition-Names

A condition-name is the name assigned to a specific value, set of values, or range of values, within the complete set of values that a data-name may assume. The data-name itself is called a conditional variable.

For example, consider a conditional variable called TITLE. If the condition-names ANALYST, PROGRAMMER, and CODER are assigned the values 1, 2, and 3 respectively, the conditional statement:

```
IF CODER .....
```

would generate a test of the value of the conditional variable TITLE against the value "3".

Procedure-Names

A procedure-name is either a paragraph-name or a section-name. Procedure-names permit one procedure to refer to others. A procedure-name may be composed solely of numeric characters. However, two numeric procedure-names are equivalent if and only if they are composed of the same number of numeric digits and have the same numeric value. Thus, 0023 is not equivalent to 23.

Literals

A literal is an item of data whose value is implied by the set of characters comprising the literal. Every literal belongs to one of two types, non-numeric and numeric, and the way in which the value is implied depends on the type. The two types are distinguished in the program by the fact that non-numeric literals must be bounded by quotation marks and numeric literals must not.

A non-numeric literal is defined as a string of any allowable characters (excluding the quotation mark), bounded by quotation marks. The value of a non-numeric literal is the string of characters itself, excluding the quotation marks. Any spaces enclosed in the quotation marks are part of the non-numeric literal and therefore part of the value. At most 132 characters may appear in a non-numeric literal.

If every character in a non-numeric literal is either one of the letters, A through Z, or is a space, the non-numeric literal has CLASS ALPHABETIC. All other non-numeric literals have CLASS ALPHANUMERIC.

Numeric Literal

A numeric literal is defined as a string of characters chosen from the numerals 0 through 9, the plus (+), the minus (-), and the decimal point. The value of a numeric literal is the algebraic quantity represented by the characters in the numeric literal.

Every numeric literal has CLASS NUMERIC. Detailed rules for the formation of numeric literals are:

1. A numeric literal must contain at most one sign character and/or one decimal point.
2. The literal must contain at least one digit.
3. The sign in the literal must appear as the left-most character of the literal. If the literal is unsigned, the literal is considered to be positive.
4. The decimal point may appear anywhere within the literal except as the right-most character of the literal, and is treated as an implied decimal point. If the literal contains no decimal point, the literal is considered to be an integer.

If a literal conforms to the rules for formation of numeric literals, but it is enclosed in quotation marks, it is considered as a non-numeric literal and will be treated as such by the compiler. In other words;

-125.65 is not the same as "-125.65".

Examples of non-numeric literals are:

1. "EXAMINE CLOCK NUMBER"
2. "PAGE 144 MISSING"
3. "-125.65"

Examples of numeric literals are:

1. 1506798
2. -12572.6
3. +256.75
4. 1435.89

A NUMERIC literal may have a leading sign, or no sign. There may be "i" integer digits and "f" fractional digits where $i + f$ is less than 19. A decimal point may appear in the literal.

Figurative Constants

Certain constants have been assigned fixed data-names. These constants with the fixed data-names are called "figurative constants." These data-names, when used as figurative constants, must not be bounded by quotation marks. If these data-names were bounded by quotation marks they would be considered as non-numeric literals. The singular and plural form of figurative constants are equivalent, and may be used interchangeably. The fixed data-names and their meanings are as follows:

{ ZERO
ZEROS
ZEROES }

Represents the value 0, or one or more occurrences of the character 0, depending on the context.

{ SPACE
SPACES }

Represents one or more blanks or spaces.

{ UPPER-BOUND
UPPER-BOUNDS }

Represents one or more of the character Z, which is conventionally used as a high delimiter in processing data.

{ LOWER-BOUND
LOWER-BOUNDS }

Represents one or more of the character 0, which is conventionally used as a low delimiter in processing data.

HIGH-VALUE
HIGH-VALUES

Represents one or more of the character which has the highest value in the computer's collating sequence.

LOW-VALUE
LOW-VALUES

Represents one or more of the character 0, which has the lowest value in the computer's collating sequence.

QUOTE
QUOTES

Represents one or more of the character ". The word QUOTE can not be used in place of a quotation mark in the Source Program to bound a non-numeric literal. Thus QUOTE ABC QUOTE is incorrect as a way of stating the non-numeric literal "ABC".

ALL Literal

Represents one or more of the string "literal". This string must be either a non-numeric literal, or a figurative constant other than "ALL Literal". When a figurative constant is used, the word ALL is redundant and is used for readability only.

The SIZE of a figurative constant is determined by the compiler from context according to the following rules:

1. When a figurative constant is associated with another data item, the string of characters specified by the figurative constant is repeated on the right until its size is equal to the size in characters of the associated data item.
2. When a figurative constant appears in a DISPLAY, EXAMINE or STOP statement, the length of the string is one character. ALL literal may not be used with DISPLAY, EXAMINE or STOP.

EXAMPLES:

1. MOVE ALL "4" TO COUNT-FIELD, where COUNT-FIELD has been described as having 6 characters, results in 444444.
2. MOVE ALL "NO-OP" TO EMPTY-AREA, where EMPTY-AREA has been described as having 12 characters, results in NO-OPNO-OPNO.
3. From the statement MOVE SPACES TO TITLE-BOUNDARY, the compiler will create coding which puts as many space characters into the item TITLE-BOUNDARY as room allows.
4. DISPLAY QUOTE, "NAME", QUOTE results in "NAME".
5. MOVE QUOTE TO AREA-A, where AREA-A has been described as having 5 characters, results in """""".

Special Register

TALLY

TALLY is the name of a special register whose size is equivalent to a five decimal digit integer. Its primary use is to hold information produced by the EXAMINE verb. It may also be used, however to hold information produced elsewhere in a program anywhere the format allows an elementary integral data item.

The description of TALLY is implicitly:

77 TALLY; SIZE IS 5 COMPUTATIONAL-1 DIGITS.

Mnemonic-Names

A mnemonic-name is a special external-name formulated according to the rules for data-names. They are associated with special I/O functions and do not have data descriptions in the Data Division. They are defined in the Special-Names sentence of the Environment Division.

Verbs

A verb is a single word which appears in the PROCEDURE DIVISION, and designates an action:

ADD, MOVE, GO, etc.

Reserved Words

Reserved words are used for syntactical purposes and may not appear as user-defined nouns or verbs. There are three types:

1. Connectives

Connectives are words used to:

- a. Denote the presence of a qualifier: OF, IN.
- b. Form compound conditionals: AND, OR, AND NOT, OR NOT; these are called logical connectives.

2. Optional Words

Optional words have been defined and used to improve the readability of the language. Within each format upper case words which are not underlined are designated as optional. The presence or absence of each optional word within the format for which it is shown does not alter the compiler's translation. However, misspelling of an optional word or its replacement by another word of any kind is not allowed.

3. Key Words

Key words are of three types:

- a. Verbs: ADD, READ, ENTER, etc.
- b. Required words, appearing in formats in various divisions of the language, needed to complete the meaning of certain verbs or entries: TO, OMITTED, MEMORY, etc.
- c. Words not shown in any format, but which have a specific functional meaning: NEGATIVE, SECTION, TALLY, etc.

Qualifiers

Every name used in a COBOL source program must be unique, either because no other name has the identical spelling, or because the name exists within a hierarchy of names (in the DATA or PROCEDURE DIVISION), such that the name can be made unique by mentioning one or more of the higher levels of the hierarchy. The higher level names are called qualifiers when used in this way, and the process is called qualification. Enough qualification must be mentioned to make the name unique, but it is not necessary to mention all levels of the hierarchy unless they are needed to make the name unique. A file-name is the highest level qualifier available for a data-name. A section-name is the highest and the only qualifier available for a paragraph-name. Thus, file-names and section names must be unique in themselves and cannot be qualified. Regardless of the available qualification, no name can be used both as a data-name and as a procedure-name.

Qualification in COBOL is performed by appending one or more prepositional phrases, using IN or OF. The choice between IN or OF is based on readability, because they are logically equivalent. Nouns must appear in ascending order of hierarchy with either of the words IN or OF separating them. The qualifiers are considered part of the name. Thus, whenever a data item or procedure paragraph is referenced, any necessary qualifiers must be written as part of the name.

Consider two records called MASTER and NEW-MASTER, with the following partial data descriptions (see Chapter VI for detailed Record Descriptions):

1	MASTER.....	1	NEW-MASTER.....
2	CURRENT-DATE...	2	CURRENT-DATE...
3	MONTH...	3	MONTH...
3	DAY...	3	DAY...
3	YEAR...	3	YEAR...
2	LAST-TRANSACTION-DATE...	2	LAST-TRANSACTION-DATE...
3	MONTH...	3	MONTH...
3	DAY...	3	DAY...
3	YEAR...	3	YEAR...

The MONTH contained in CURRENT-DATE of NEW-MASTER must be referred to as:

MONTH { $\frac{IN}{OF}$ } CURRENT-DATE { $\frac{IN}{OF}$ } NEW-MASTER

while the DAY of the LAST-TRANSACTION-DATE of the MASTER record must be referred to as

DAY { $\frac{IN}{OF}$ } LAST-TRANSACTION-DATE { $\frac{IN}{OF}$ } MASTER

Note that it is permissible to use IN or OF interchangeably.

The following rules must be obeyed in using qualifications:

1. A qualifier must be of a higher level and within the same hierarchy as the name it is qualifying.

2. The same name may not appear at two levels in a hierarchy so that it would appear to qualify itself.
3. If a data-name or condition-name is assigned to more than one data item in a program, it must be qualified in all references to it in the PROCEDURE DIVISION and the ENVIRONMENT DIVISION, and in all references to it in defining clauses except REDEFINES in the DATA DIVISION.
4. Any data-name requiring qualification, must be qualified every time it is referenced, i.e., the absence of qualification is not considered qualification.
5. A paragraph-name must not be duplicated within the same section. A paragraph-name can only be qualified by a section-name. When it is, the word SECTION must not appear. A paragraph-name need not be qualified when referred to from within the same SECTION. Subscripts and conditional variables, as well as procedure and data-names, may be made unique by qualification where necessary or desirable.
6. A data-name cannot be subscripted when it is being used as a qualifier.
7. A name can be qualified even though it does not need qualification. The use of more names for qualification than are actually required for uniqueness is permitted. If there is more than one combination of qualifiers which ensure uniqueness, then any set can be used.
8. The name of a conditional variable can be used as a qualifier for any of its condition-names.

Subscripts

The technique of subscripting is most commonly used for table-handling functions. The ability to reference individual elements (of a table or list) which have not been assigned individual data-names is provided by using subscripts; the ability to reference the entire table or list is provided by using the name of the table or list.

A subscript is an integer whose value determines which element is being referred to within a table (or list) of like elements. The subscript may be represented either by a literal which is an integer (e.g., 25), or by a data-name (e.g., AGE) which is a numeric elementary data item whose data description must not include any character positions to the right of the assumed decimal point.

When the subscript is represented by a data-name, the data-name must be described by a Record Description entry in the DATA DIVISION. In both cases, i.e., whether the subscript is represented by a literal or a data-name, the subscript is enclosed in parentheses, and appears immediately after the terminal space of the name of the element referenced, e.g., RATE (AGE) or RATE (25).

Tables are often defined so that more than one level of subscripting is required to locate an element within them. A maximum of three levels of subscripting is permitted by COBOL. Multilevel subscripts are always written from left to right in the order: major, intermediate, minor. In this case, the subscripts are shown in a single pair of parentheses, and separated by commas. For example:

```

RATE (REGION, STATE, CITY)
RATE (3, STATE, CITY)
RATE (3, 5, 6)

```

All of the above would reference a particular rate in a three-dimensional table of rates.

A subscript value of "1" denotes the first element of a list, a value of "2", the second element, etc. A subscript of (1,2) denotes the second element within the first repeated element of the table. A table with its main element appearing 10 times, its intermediate element appearing 5 times within each of the major elements, and the minor element appearing 3 times within each of the intermediate elements, is considered a three-dimensional table. The last element of such a table is referenced by the use of the subscript (10,5,3).

No element of a table or list may be referenced without a subscript. However, the entire table may be referenced, provided the table has been given a unique name. Reference to a data-name within a table or list must include all subscripts upon which the data-name is dependent. Use of more than, or less than, the correct number of subscripts is considered an error. Some examples of the writing of subscripts are:

```

MOVE RATE (AGE) TO LISTING.
IF HEIGHT (10) IS GREATER THAN.....
MULTIPLY PRICE (STOCK-NO) BY INVENTORY (STOCK-NO) .
EXAMINE CLASS (REGION) REPLACING.....
MOVE RATE-TABLE TO OUTPUT-AREA...

```

If references to a conditional variable require subscripting, then references to its condition-names also require subscripting.

If a data item is repeated (i.e., involves the OCCURS clause at its own or higher level) then the name of this item must be subscripted whenever it is referenced. Furthermore, a data-name can only be subscripted if the data item is repeated.

Regardless of the above rules, a data-name must not be subscripted under any of the following conditions:

1. Where the data-name is being used as a subscript.
2. Where the data-name is being used as a qualifier.
3. When the data-name appears in the defining clauses in the DATA DIVISION.

DISTINCTION BETWEEN SUBSCRIPTS AND QUALIFIERS

There is a distinct difference between qualification and subscripting. Qualification is necessary when the same data-name is used for several different items of data; subscripting is necessary when some of the elements of a table or list have not been assigned individual names.

In subscripting a data-name the following rules (which have already been specified above) are significant:

1. The qualifiers are considered part of the data-name.
2. A data-name being used as a qualifier cannot be subscripted.
3. Qualification not logically required for uniqueness is permitted.

As a result of these rules there are several correct ways of expressing the subscripted data-names. For example, if a data item named A occurs 5 times and contains a data item named B which occurs 4 times in each A, and each B, in turn, contains a data item C which occurs twice in each B, then the following expressions are all correct references to the last C, i.e., to the 2nd C in the 4th B in the 5th A:

```
C IN B IN A (5,4,2)
C IN B (5,4,2)
C IN A (5,4,2)
C (5,4,2)
```

The first three of these are examples of unnecessary, although permissible, qualifications assuming that C and B only occur in this hierarchy. However, if the name C is used elsewhere, then the qualification must be used. Note that the following forms of expression are incorrect:

```
C (5,4,2) IN B IN A
C (2) IN B (4) IN A (5)
C (4,2) IN A (5)
C (2) IN B (5,4)
```

Series Connectives

When two or more nouns are written in a series, words or characters may be used as connectives between the nouns. The use of such connectives in a series of nouns is optional, unless the nouns represent conditions and require logical connectives. The connectives which may be used are:

```
, AND
,
, OR
AND
OR
```

COMPLETE LIST OF RESERVED WORDS

The words shown are part of the COBOL System. Users must avoid using these words for data- or procedure-names.

ABOUT	COMMA	ENDING
ACCEPT	COMMERCIAL	ENDING-FILE-LABEL
ACCESS	COMMON	ENDING-TAPE-LABEL
ACTUAL	COMP	END-OF-FILE
ADD	COMP-1	END-OF-TAPE
ADDRESS	COMP-2	ENTER
ADVANCING	COMP-3	ENTRY
AFTER	COMPILE	ENVIRONMENT
ALL	COMPUTATIONAL	EQUAL(S)
ALPHABETIC	COMPUTATIONAL-n	ERROR(S)
ALPHANUMERIC	COMPUTE	EVERY
ALTER	CONSTANT	EXAMINE
ALTERNATE	CONFIGURATION	EXCEEDS
AN	CONSOLE	EXIT
AND	CONTAINS	EXPONENTIATED
APPLY	CONTROL(S)	
ARE	COPY	FD
AREA(S)	CORR	FILE
ASCENDING	CORRESPONDING	FILE-CONTROL
ASSIGN	CURRENCY	FILE-LIMIT(S)
AT	DATA	FILE-SERIAL-NUMBER
AUTHOR	DATE-COMPILED	FILLER
	DATE-WRITTEN	FILLING
BCD	DE	FINAL
BEFORE	DEBUG	FIRST
BEGINNING	DECIMAL	FLOAT
BEGINNING-FILE-LABEL	DECIMAL-POINT	FOOTING
BEGINNING-TAPE-LABEL	DECLARATIVES	FOR
BINARY	DEFINE	FORMAT
BIT(S)	DEFINITIONS	FROM
BLANK(S)	DENSITY	
BLOCK	DEPENDING	GE-625
BLOCK-COUNT	DESCENDING	GE-635
BY	DETAIL	GEIN
	DIGIT(S)	GELAPS
CALL	DISPLAY	GENERATE
CARD(S)	DISPLAY-n	GETIME
CF	DIVIDE	GIVING
CH	DIVIDED	GMAP
CHARACTER(S)	DIVISION	GO
CHECK	DOLLAR	GREATER
CLASS	DOWN	GROUP
CLOCK-UNITS		
CLOSE		
COBOL	ELECT	HASHED
CODE	ELSE	HEADING
COLLATE	END	HIGH
COLUMN		HIGH-VALUE(S)

I-O-CONTROL	MULTIPLIED	PUNCH
ID	MULTIPLY	PURGE-DATE
IDENTIFICATION		
IF	NEGATIVE	QUOTE
IN	NEXT	RANDOM
INCLUDE	NO	RANGE
INDEX	NO-MEMORY-DUMP	RD
INDEXED	NOT	READ
INDICATE	NOTE	READER
INITIAL	NUMBER	RECORD(S)
INITIATE	NUMERIC	RECORD-COUNT
INPUT		RECORDING
INPUT-OUTPUT	OBJECT-COMPUTER	REDEFINES
INSTALLATION	OBJECT-PROGRAM	REEL
INTO	OCCURS	REEL-NUMBER
IS	OF	REEL-SERIAL-NUMBER
	OFF	RELEASE
JUST	OH	REMARKS
JUSTIFIED	OMITTED	RENAMES
	ON	RENAMING
KEY	ONLY	REPLACING
	OPEN	REPORT(S)
LABEL	OPTIONAL	REPORTING
LABEL-DAY	OPTIONS	RERUN
LABEL-IDENTIFIER	OR	RESET
LABEL-YEAR	OTHERWISE	RESERVE
LAST	OUTPUT	RETENTION-PERIOD
LEADING	OV	RETURN
LEAVING	OVERFLOW	REVERSED
LEFT	OVERLAY	REWIND
LESS		RF
LEVEL	PAGE	RH
LIBRARY	PAGE-COUNTER	RIGHT
LIMIT(S)	PERFORM	ROUNDED
LINE(S)	PF	RUN
LINE-COUNTER	PH	
LINKAGE	PHASE1	SAME
LISTING	PIC	SD
LOCATION	PICTURE	SEARCH
LOCK	PLACE(S)	SECTION
LOW	PLUS	SECURITY
LOW-VALUE(S)	POINT	SEGMENT-LIMIT
LOWER-BOUND(S)	POPUP	SELECT
	POSITION	SELECTED
MAGNETIC	POSITIVE	SENTENCE
MEMORY	PREPARED	SENTINEL
MEMORY-DUMP	PRIORITY	SEQUENCED
MEMORY-DUMP-KEY	PROCEDURE	SEQUENTIAL
MINUS	PROCEED	SERIAL
MODE	PROCESS	SET
MODULES	PROCESSING	SIGN
MOVE	PROGRAM	SIGNED
MULTIPLE	PROGRAM-ID	SIZE
	PROTECT	SORT
	PROTECTION	

SOURCE	WHEN
SOURCE-COMPUTER	WITH
SPACE(S)	WORDS
SPACE-SAVING	WORKING-STORAGE
SPECIAL-NAMES	WRITE
SPEED	
STANDARD	ZERO
STATEMENTS	ZEROES
STATUS	ZEROS
STORAGE	
STOP	
SUBTRACT	
SUM	
SUPERVISOR	
SUPPRESS	
SWITCH	
SYMBOL	
SYNC	
SYNCHRONIZED	
SYSOUT	
TALLY	
TALLYING	
TAPE	
TERMINATE	
TEST-PATTERN	
TIME-SAVING	
TIMES	
THAN	
THEN	
THROUGH	} Equivalent
THRU	
TIME(S)	
TO	
TOP	
TYPE	
TYPEWRITER(S)	
UNEQUAL	
UP	
UPPER-BOUND(S)	
UNIT(S)	
UNTIL	
UPON	
USAGE	
USE	
USING	
VLR	
VALUE(S)	
VARYING	

VI. DATA DIVISION

GENERAL DESCRIPTION

Data to be processed falls into three categories:

1. That which is contained in files and enters or leaves the internal memory of the computer from a specified area or areas.
2. That which is developed internally and placed into intermediate or working storage, or placed into specific formats for output reporting purposes.
3. Constants which are defined by the user. (Figurative constants and literals used in procedure statements are not listed in the DATA DIVISION.)

The approach taken in defining file information is to distinguish between the physical aspects of the file (the File Description) and the conceptual characteristics of the data contained therein (the Record Description).

Physical aspects are defined as:

1. The mode in which the file is recorded.
2. The grouping of logical records within the physical limitations of the file medium.
3. The means by which the file can be identified.

The conceptual characteristics are the explicit definitions of each logical entity within the file itself.

For purposes of processing, the contents of a file are divided into logical records. By definition, a logical record is any consecutive set of information. For example: in an Inventory Transaction File, a logical record could be defined as a single transaction, or as all consecutive transactions which pertain to the same stock item. It is important to note that several logical records may occupy a physical record.

The concept of a logical record is not restricted to file data, but is carried over into the definition of working storages and constants. Thus, working storages and constants may be grouped into logical records and defined by a series of Record Description entries.

ORGANIZATION

The DATA DIVISION is subdivided according to types of data. It consists of a FILE Section, a WORKING-STORAGE Section, a CONSTANT Section, and a REPORT Section, written in that order.

The FILE Section contains File Descriptions, Sort-file Descriptions, and Record Descriptions for both label and data records in files, and for data records in sort-files. Label records and data records are defined in the same manner; however, because the input/output system of the object program must perform special operations on label records, fixed names have been assigned to certain label records items on which such operations must be performed. All Record Description entries pertaining to label records and data records of a file must immediately follow the File Description sentence. The File and Sort-file Description entries represent the highest level of organization in the FILE Section.

FILE DESCRIPTION ENTRY

A File Description entry contains information pertaining to the physical aspects of a file. In general, it may include the following:

1. The manner in which the data is recorded on the file.
2. The size of the logical and physical records.
3. The names and values of the label records contained in the file.
4. The names of the data records which comprise the file.
5. The keys on which the data records have been sequenced.

The listing of data and label record names in a File Description entry serves as a cross reference between the file and records in the file.

A Sort-file Description can be considered to be a particular type of File Description. A file is a set of records, coming in from or going out to an external medium such as magnetic tape, which has certain label conventions and which has certain rules of physical blocking. The records of a file are assumed to be processed one at a time.

A Sort-file is a name for the set of records to be sorted by a SORT statement. Records are brought into this set by being RELEASED to the sort-file during the INPUT PROCEDURE specified by the SORT statement. At the conclusion of the INPUT PROCEDURE, the set of records which have been RELEASED to the sort-file constitute the whole sort-file. There are no label procedures which the user can control, and the rules for blocking and internal storage are peculiar to the SORT verb. The SORT verb rearranges the entire set of records in the sort-file according to the keys specified in the SORT statement. Each of the sorted records can be made available, in order, by being RETURNed from the sort-file during the output procedures specified by the SORT statement.

The RELEASE and RETURN statements imply nothing with respect to buffers, blocks, or reels. The sort-file created by the execution of RELEASE statements can only be SORTed and its records can only be obtained by being RETURNed from the sort-file during the OUTPUT PROCEDURE. The sort-file cannot be OPENed or CLOSED or used in any other way.

In a Sort-file Description, only information about the size and number of data records can be given.

ENTRY FORMATSGeneral Notes

A File (Sort-file) Description entry consists of a level indicator, a file (sort-file) name, and a series of independent clauses which define the physical and logical characteristics of the file (sort-file). The mnemonic level indicator FD (SD) is used to identify the start of a File (Sort-file) Description entry. In this manner the File (Sort-file) Description entries are distinguished from those associated with a Record Description.

Specific Formats

The individual clause formats are arranged in an alphabetic order.

File Description Complete Entry

FUNCTION: To furnish information concerning the physical structure, identification and record types of a file.

Option 1:

FD file-name COPY library-name.

Option 2:

FD file-name [; RECORDING MODE IS { BINARY } { { HIGH } } DENSITY]]
 [; FILE CONTAINS ABOUT integer-1 RECORDS]
 [; BLOCK CONTAINS [integer-2 TO] integer-3 { RECORDS }]
 [; RECORD CONTAINS [integer-4 TO] integer-5 CHARACTERS]
 ; LABEL { RECORDS ARE } { STANDARD }
 { RECORD IS } { OMITTED }
 [; VALUE OF data-name-1 IS literal [,data-name-2 IS...]]
 { { ; DATA { RECORD IS }
 { RECORDS ARE } } data-name-3 [, data-name-4 ...]
 { { ; REPORT IS }
 { ; REPORTS ARE } }
 ; DATA { RECORD IS } ... { ; REPORT IS }
 { RECORDS ARE } ... { ; REPORTS ARE } ... }
 [; SEQUENCED ON data-name-5 [, data-name-6...]]

Option 3:

SD file-name COPY library-name.

Option 4:

SD file-name [; FILE CONTAINS ABOUT integer-1 RECORDS]
 [; RECORD CONTAINS [integer-2 TO] integer-3 CHARACTERS]
 ; DATA { RECORD IS } data-name-1 [, data-name-2 ...] .
 { RECORDS ARE }

1. The level indicator FD or SD identifies the beginning of the file or sort description respectively and precedes the file-name. (See Chapter III.)
2. The clauses which follow the name of the file are optional in many cases. For further details, see the individual explanations for each clause.
3. Those entries which require more than a single line are continued on subsequent lines with the same left margin for each line. That is, with each subsequent line starting under the first letter in the file-name. (See the Reference Format for the DATA DIVISION, Chapter III.)
4. If the level indicator SD is used, the file-name must be the name associated with a sort-file. The file-name can appear in the PROCEDURE DIVISION (other than as a qualifier) only in SORT and RETURN statements. The file-name must never appear in a READ, WRITE, USE, OPEN or CLOSE statement.
5. When the level indicator is SD and more than one record is present for the Sort File, the first record determines the "dominant record size" parameter for the Sort (see GE-625/635 SORT/MERGE, CPB-1005). For further information on sort record descriptions see Chapter VII.

BLOCK Size

FUNCTION: To specify the size of a physical record (or block).

[; BLOCK CONTAINS [integer-2 TO] integer-3 { CHARACTERS
RECORDS
RECORD }]

Notes:

1. This clause may be omitted when the file is assigned to magnetic tape and has the standard physical record size (320 computer words).
2. Whether the block size is stated in terms of characters or records:
 - a. Each logical record begins in a new computer word.
 - b. No logical record may be longer than a physical record.
 - c. Each physical record contains an integral number of logical records.
3. When the CHARACTERS option is used, the physical record size is specified in terms of the number of standard characters contained within the physical record, regardless of the types of characters used to represent the items within the physical record.
4. If only integer-3 is shown then it represents the exact size of the physical record. If integer-2 and integer-3 are both shown, then they refer to the minimum and maximum size of the physical record respectively. In that case, integer-2 is understood to be documentation only.
5. The word CHARACTERS within the BLOCK clause is an optional word. Whenever the key word RECORD(S) is not specifically written in the BLOCK clause, integer-2 and integer-3 represent CHARACTERS. The RECORD(S) option must not be used for a file for which REPORT(S) are specified.
6. When the RECORDS option is used with variable-length records, the BLOCK size is equal to the maximum record size, in computer words, multiplied by the number of records plus 1.

COPY

FUNCTION: To obtain a file description entry or sort file description entry from the COBOL library.

$\left\{ \begin{array}{c} \text{FD} \\ \text{SD} \end{array} \right\}$ data-name COPY library-name.

Notes:

1. COPY is used when the COBOL library contains the File Description entry or the Sort File Description entry.
2. During compilation, the COPY clause is replaced by the sequence of clauses that follows the level indicator (FD or SD) and file-name within the library-name entry.
3. File code must be .L for COBOL library.

DATA RECORDS

FUNCTION: To cross reference the description of data records with their associated file.

; DATA { RECORD IS
RECORDS ARE } data-name-6 , [data-name-7 ...]

Notes:

1. Either this clause or the REPORT clause is required in every File Description entry.
2. The presence of more than one data-name indicates that the file contains more than one type of data record. If record sizes (in words) are unequal, the file is assigned the Variable-Length Record format, and RECORDING MODE must be BINARY (explicitly or implicitly). The records may be of differing sizes, formats, etc. The order in which they are listed is not significant except for sort files (see note 5, below).
3. Conceptually, all data records within a file share the same area. This is in no way altered by the presence of more than one type of data record within the file.
4. Both data-name-6 and data-name-7 must have 01 level numbers. Subscripting of these data-names is not permitted, and qualification is not necessary since they are implicitly qualified by the file-name of this entry.
5. For a sort-file with more than one data record described, the first record mentioned in the DATA RECORDS clause is assumed to be the dominant type; its size is considered to be the most prevalent in the sort-file. Sort optimization is based upon this assumption, so a careful choice will favor object-program efficiency.

FILE Size

FUNCTION: To indicate the approximate number of logical records in a file.

[; FILE CONTAINS ABOUT integer-1 RECORDS]

Notes:

1. This optional clause may be used for documentation.

LABEL RECORDS

FUNCTION: To cross reference the descriptions of label records with their associated file.

; LABEL { RECORDS ARE } { STANDARD }
 { RECORD IS } { OMITTED }

Notes:

1. This clause is required for all files. When a file contains no labels, the word OMITTED must be used.
2. The following four types of label records may appear on the tapes associated with a file. Since the type of label is significant, the fixed record-names shown in capital letters have been assigned:
 - a. A BEGINNING-TAPE-LABEL which appears at the beginning of each tape and which precedes all other information, and contains information about the tape.
 - b. A BEGINNING-FILE-LABEL which appears only once in a file, and precedes the first data record in the file. This label contains information about the file.
 - c. An ENDING-TAPE-LABEL which immediately follows the last valid data or label record on the tape. The label must appear before the physical end of the tape is encountered. This label may contain information about the tape.
 - d. An ENDING-FILE-LABEL which appears only once in a file and which immediately follows the last data record on the last reel of a file, and may contain information about the file.
3. The formats of STANDARD label records are understood by the compiler, and need not be described in the source-program. All other types of labels must be defined as data rather than label records. All such records must be listed in the "DATA RECORDS" clause rather than in the "LABEL RECORDS" clause.
4. On a Multifile tape, either all files must be labeled (LABEL RECORDS ARE STANDARD), or else none may be labeled (LABEL RECORDS ARE OMITTED).

RECORD Size

FUNCTION: To specify the size of data records.

[; RECORD CONTAINS [integer-4 TO] integer-5 CHARACTERS]

Notes:

1. The size of each data record is completely defined within the Record Description entries; therefore this clause is never required.
2. Integer-5 may not be used by itself unless all the data records in the file have the same size. In this case integer-5 represents the exact number of characters in the data record. If integer-4 and integer-5 are both shown, they refer to the minimum number of characters in the smallest size of data record and the maximum number of characters in the largest size data record, respectively.
3. The size is specified in terms of the number of standard characters contained within the logical record, regardless of the types of characters used to represent the items within the logical record. The size of the records are determined according to the rules for determining the size of a group item.

RECORDING MODE

FUNCTION: To specify the format of data on external media.

[; RECORDING MODE IS { BINARY
BCD } [{ HIGH
LOW } DENSITY]]

Notes:

1. This clause is interpreted only for magnetic tape files. If it is omitted on a tape file, the recording mode is understood to be BINARY HIGH DENSITY.
2. The HIGH or LOW DENSITY option may be used for documentation purposes only if the file is assigned to other than magnetic tape.
3. If data-names within the file have USAGE COMPUTATIONAL[-n] and/or DISPLAY-2, the RECORDING MODE must be BINARY. The same rule applies to any file containing REPORT[S].

REPORT(S)

FUNCTION: To cross reference the description of Report Description entries with their associated File Description entry.

{ REPORT IS
REPORTS ARE } data-name-6 [, data-name-7 ...]

Notes:

1. This clause is required in the File Description entry if the file is an output report file.
2. The presence of more than one data-name indicates that the file contains more than one report. These reports may be of differing sizes, differing formats, etc. The order in which they are listed is not significant.
3. Each data-name listed in the FD entry must have a RD (Report Description) entry in the Report Section.

SEQUENCED

FUNCTION: To indicate the keys on which data records are sequenced.

[; SEQUENCED ON data-name-8 [, data-name-9 ...]]

Notes:

1. This optional clause may be used for documentation, and does not result in object-program action. It does not result in an automatic sequence check.
2. Data-name-8 represents the major key, data-name-9 represents the next highest key, etc.
3. The data-names should be qualified when necessary, but subscripting is not permitted.

VALUE

FUNCTION: To specify the actual value of items which appear in label records.

[VALUE OF data-name-3 IS literal-1 [, data-name-4 IS literal-2]]

Notes:

1. Data-name-3 and data-name-4 can only be the fixed label item names IDENTIFICATION and RETENTION-PERIOD. Either or both may be given.
2. If IDENTIFICATION is specified, the action in the object-program depends upon the use of the file. For an input file, the object-program's label check routine verifies that the value of the IDENTIFICATION in beginning labels of the file is equal to the given literal. For an output file, the literal is implicitly moved to IDENTIFICATION before a beginning label is written. The literal associated with IDENTIFICATION must be a non-numeric literal of no more than 12 characters.
3. RETENTION-PERIOD has no significance for input files. For an output file, the given literal is implicitly moved to RETENTION-PERIOD before a beginning label is written. The literal associated with RETENTION-PERIOD must be a positive integer not exceeding 999. The value 999 signifies permanent retention.

If a tape to be used for output has a RETENTION-PERIOD given in its beginning label, the object-program's output routine checks that value against the current date to assure that the RETENTION-PERIOD has expired before it begins writing new data on the tape.

RECORD DESCRIPTION

A Detailed Data Description consists of a set of entries. Each entry defines the characteristics of a particular unit of data. With minor exceptions, each entry is capable of completely defining a unit of data. Because the COBOL Detailed Data Descriptions involve a hierarchical structure, the contents of an entry may vary considerably, depending upon whether or not it is followed by subordinate entries.

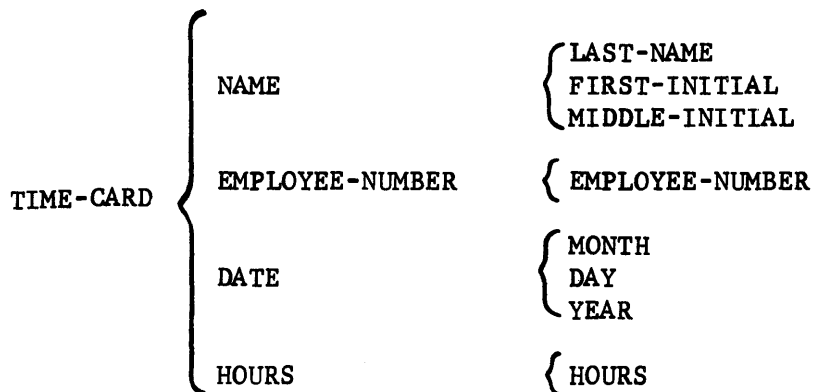
In defining the lowest level or subdivision of data, the following information may be required:

1. A level number which shows the relationship between this and other units of data.
2. A data-name.
3. The SIZE in terms of the number of Standard Data Format characters.
4. The dominant USAGE of the data.
5. The number of consecutive occurrences (OCCURS) of elements in a table or list.
6. The RANGE of values which the data may assume.
7. The CLASS or type of data - i.e., ALPHABETIC, NUMERIC or ALPHANUMERIC.
8. The presence of an operational SIGN.
9. Location of an actual or an assumed radix point.
10. Location of editing symbols such as dollar signs and commas.
11. Justification and synchronization of the data. (JUSTIFIED, SYNCHRONIZED.)
12. Special editing requirements such as zero suppression and check protection.
13. Initial VALUE of a working-storage item or fixed VALUE of a constant.

An entry which defines a unit of data must not be contradicted by a subordinate entry. Thus, once the CLASS is defined, it applies to all subordinate entries and need not be re-specified in the subordinate entries. However, when CLASS is defined as ALPHANUMERIC, subordinate entries may particularize the class by specifying ALPHABETIC or NUMERIC. If the CLASS has been defined as either ALPHABETIC or NUMERIC, however, subordinate entries may not change the CLASS.

Concept of Levels

A level concept is inherent in the structure of a logical record. It arises in a natural way from the need to specify subdivisions of a record for the purpose of data reference. Once a subdivision has been specified, it may be further subdivided to permit more detailed data referencing. For example, a weekly TIME-CARD record might be divided into four major items: NAME, EMPLOYEE-NUMBER, DATE and HOURS, with more specific information on DATE and NAME as follows:



The most basic subdivisions of a record, that is, those not further subdivided, are called elementary items; consequently, a record is said to consist of a sequence of elementary items, or the record itself may be an elementary item.

Often it is desirable to reference a set of elementary items - particularly with the MOVE verb. For this reason, elementary items may be combined into groups, each group consisting of a sequence of one or more elementary items. Groups, in turn, may be combined into groups of two or more groups, etc. The term "item", in future discussions, denotes either an elementary item or a group.

A system of level numbers is employed in COBOL to show the organization of elementary items and groups. Level numbers start at 01 for records, since records are the most inclusive groups possible. Less inclusive groups are assigned higher (not necessarily successive) level numbers not greater in value than 49. (NOTE: There are "special" level numbers, 66, 77, and 88, discussed below, which are exceptions to this rule.) Separate entries are written in the source program for each level.

Using the TIME-CARD example above, a skeleton source program listing, showing the use of level numbers to indicate the hierarchical structure of the data, might appear as follows:

```

01 TIME-CARD
  04 NAME
    6 LAST-NAME
    06 FIRST-INITIAL
    06 MIDDLE-INITIAL
  04 EMPLOYEE-NUMBER
  04 DATE
    05 MONTH
    5 DAY
    05 YEAR
  04 HOURS

```


For the sake of simplicity, only the level number and data-name of each entry have been given in the above example. A complete description would, of course, have included information on SIZE, CLASS, USAGE, etc.

A group includes all groups and elementary items described under it until a level number less than or equal to the level number of that group is encountered. Thus, in the above example, HOURS is not a part of the group called DATE. MONTH, DAY and YEAR are a part of the group called DATE, because they are described immediately under it and have a higher level number.

It should be noted that an elementary item may belong to more than one group. In the previous example, the elementary item called YEAR belongs to the group called DATE, and also to the group called TIME-CARD. A more detailed illustration of this point is given in the following example:

```

01  ABLE
    03  BAKER
        04  CHARLIE
        04  DOG
    03  EASY
        04  FOX
    03  GEORGE
        08  HOW
        09  IVY

```

Elementary item IVY belongs to groups HOW, GEORGE and ABLE. FOX belongs to groups EASY and ABLE, while CHARLIE and DOG belong to BAKER and ABLE.

The level number of an entry (elementary item or group) immediately following the last elementary item of a previous group, must be that of one of the groups to which that elementary item belongs. Application of this rule to the previous example restricts the level number of EASY to either 1 or 3, since these are the level numbers of the groups containing DOG. Specifically, EASY may not have the level number 2. That is, the following example is incorrect:

```

01  ABLE
    03  BAKER
        04  CHARLIE
        04  DOG
    02  EASY

```

Three types of data exist for which there is no true concept of level; namely, independent constants and working-storage items, and names introduced by a RENAME clause.

Independent constants and independent working storage items which bear no relationship to one another and which are not further subdivided are assigned the special level number 77.

Entries which specify condition-names are assigned the special level number 88.

Data-names introduced by a RENAME clause for the purpose of renaming or regrouping elementary items are assigned the special level number 66.

Concept of a Computer Independent Detailed Data Description

For a description to be computer independent, it is necessary for the subject of the description to be computer independent. A description of the format in which data is carried on a particular external medium (External Format), or within a particular computer (Internal Format), is applicable only to a limited range of compatible equipment. Therefore, the characteristics of properties of the data are described in relation to a Standard Data Format rather than an equipment oriented format. This Standard Data Format is oriented to general data processing applications.

The Standard Data Format uses the decimal system to represent numbers regardless of the radix used by the computer.

The Standard Data Format describes non-numeric elementary items using characters which are alphabetic or alphanumeric. In general, a complete detailed description will be written for elementary items only, since the characteristics of a group item are dependent upon the characteristics of the elementary items contained within it.

Derivation of External and Internal Format

External and Internal Formats of a data item are affected by characteristics of the item, the characteristics of the computer equipment, and the processing of the item throughout the system. Since the equipment is described in the File Description and ENVIRONMENT DIVISION, the detailed data description must provide the description of the data and its processing throughout the system. This allows a format to be selected by a user which will take advantage of the features of his computer.

Algebraic Signs

Algebraic signs are used for two purposes: to show whether the value of an item involved in an operation is positive or negative (i.e., an operational sign), or to identify the value of an item as positive or negative on an edited report for external use (i.e., a display sign.)

Display signs are not oriented to the equipment. Several alternative methods are provided for displaying the sign of an item (see PICTURE); it should be noted that these signs are not operational signs and will not generally be treated as the sign of the item when the item is used as a source in an operation.

Item Alignment and Spacing on Fixed Word Length Computers

Fixed word length computers can operate more efficiently on items which conform to the fixed structure of the computer word.

Items which may be subscripted are often aligned to computer words for more efficient handling. Another factor is the position which the item would

normally occupy (e.g., would the item be located in two words rather than one). Items which occupy separate computer words are defined as SYNCHRONIZED. A SYNCHRONIZED item is assumed to be introduced and carried in that form. Conversion to that form occurs only during the execution of a procedure (other than READ or WRITE) which stores data in the item.

Report Editing

Various forms of special editing are provided for the preparation of reports. In general, this editing will be performed only on elementary items and will be given special consideration only when the item is receiving data, not when it is being used as a source (i.e., items will not be de-edited).

ENTRY FORMATS

General Notes

A Record Description consists of a set of entries. Each record description entry, itself, consists of a level number, data-name, and a series of independent clauses.

Specific Formats

The individual clause formats are arranged in an alphabetic order, but the "Complete Entry Skeleton" shows the clauses appearing in the recommended order.

RECORD DESCRIPTIONComplete Entry Skeleton

FUNCTION: To specify the characteristics of a particular item of data.

Option 1:

level-number data-name ; [REDEFINES...]; COPY ...

Option 2:

level-number { data-name
FILLER } [; REDEFINES ...] [; SIZE...] [; USAGE...]
[; OCCURS ...] [; SIGNED ...] [; SYNCHRONIZED ...] [; POINT ...]
[; CLASS ...] [; PICTURE ...] [; JUSTIFIED ...] [; RANGE ...]
[; editing clauses ...] [; VALUE ...] .

Option 3:

66 data-name-1 RENAMES data-name-2 [THRU data-name-3] .

Option 4:

88 condition-name { VALUE IS
VALUES ARE } ...

Notes:

1. For a detailed explanation of the Reference Format used in the DATA DIVISION, see Chapter III.
2. Those clauses which begin with SIGNED, SYNCHRONIZED, POINT, PICTURE, JUSTIFIED, and RANGE, as well as the editing clauses, must not be specified except at the elementary item level.
3. The clauses may be written in any order with one exception. REDEFINES, when used, should immediately follow the data-name.
4. All semicolons are optional in the Record Description entry.
5. Detailed discussion of each of the options is found on succeeding pages.

CLASS

FUNCTION: To indicate the type of data being described.

[; CLASS IS { ALPHABETIC
NUMERIC
ALPHANUMERIC
AN }]

Notes:

1. AN is an acceptable abbreviation for ALPHANUMERIC.
2. The CLASS clause can be written at any level. If the CLASS clause is written at a group level, it applies to each elementary item in the group. The CLASS of an item cannot contradict the CLASS of a group to which the item belongs. ALPHABETIC or NUMERIC items within an ALPHANUMERIC group are not considered contradictory.
3. NUMERIC describes data composed of the characters 0-9 with or without an operational sign. If there is no sign associated with a NUMERIC item, the item is considered positive. If the item is NUMERIC and no assumed decimal point is indicated, the item is considered to be an integer.
4. ALPHABETIC describes data which contains any combination of the twenty-six (26) letters of the English alphabet and the space. No other characters can be used.
5. ALPHANUMERIC describes data which may contain any allowable character in the object computer's character set, including alphabets and numerics. Thus, data which is ALPHABETIC or NUMERIC is also ALPHANUMERIC.
6. If both PICTURE and CLASS are given, the class of characters shown in PICTURE must not contradict the CLASS clause of an elementary item, or of a group to which the item belongs.
7. If the CLASS of an elementary item cannot be determined from any clause in the item's Record Description or from the Record Description of any group to which the item belongs, the CLASS of the item is assumed to be ALPHANUMERIC.
8. An item's CLASS may be implied by various other clauses. If a combination of such clauses appears, either within an entry or between an entry and a group which contains it, they must not contradict each other.

- a. The CLASS is NUMERIC if any of the following clauses appears:

CLASS NUMERIC
USAGE COMPUTATIONAL
USAGE COMPUTATIONAL-n
SIZE integer NUMERIC
SIZE integer COMPUTATIONAL
SIZE integer COMPUTATIONAL-n
SIGNED
PICTURE containing no characters other than 0's, 9's, P's,
S, V.

- b. The CLASS is ALPHABETIC if any of the following clauses appears:

CLASS ALPHABETIC
SIZE integer ALPHABETIC
PICTURE containing only A's or a combination of A's and B's.

- c. The CLASS is ALPHANUMERIC if any of the following clauses appears:

CLASS ALPHANUMERIC (or AN)
SIZE integer ALPHANUMERIC (or AN)
PICTURE containing X
PICTURE containing 9 as well as either A or B
PICTURE containing both A and zero (0).
PICTURE containing A, *, \$, comma (,), decimal point (.),
-, +, CR, or DB.

COPY

FUNCTION: To duplicate within this record a description found previously in the source program or contained in a library.

level-number data-name-1 COPY data-name-2 [FROM LIBRARY] .

Notes:

1. The duplication process replaces the COPY clauses by the clauses (if any) that follow the data-name in the data-name-2 entry. The duplication process then inserts following the data-name-1 entry all record description entries that are subordinate to the data-name-2 entry, that is, up to but excluding the appearance of an entry whose level-number is equal to or less than the level-number of the data-name-2 entry, or whose level-number is 66.

If there are items subordinate to data-name-1, it is the user's responsibility to insure that the resulting hierarchical structure is correct. If the level-number of data-name-1 is 77, data-name-2 must be an elementary item. During the duplication process the level-numbers of all the inserted entries, except those whose level-number is 66 or 88, are adjusted by an amount equal to the difference between the level-numbers of data-name-1 and data-name-2. An error will be indicated by the compiler if this adjustment process gives rise to a level-number that exceeds 49.

2. If the level-numbers of data-name-1 and data-name-2 are both 1, any level 66 entries associated with the data-name-2 record description are inserted by the duplication process.
3. When the information to be duplicated is in the library, the additional clause FROM LIBRARY must be included. If the FROM LIBRARY option is used, the name of the level 1 entry in the library must be included in the qualification of data-name-2 unless data-name-2 is itself a level 1 entry. This is required even if the level 1 name is not necessary to make the reference unique.
4. A COPY clause can appear in the data-name-2 entry or in an entry that is subordinate to data-name-2 only if the FROM LIBRARY option appears with that clause.
5. Data-name-2 may be qualified but not subscripted.

Data-name

FUNCTION: To specify the name of the data item being described, or to specify an unused portion of the logical record.

{ data-name
FILLER }

Notes:

1. A data-name or the key word FILLER must be the first word following the level-number in each Record Description entry and must not be qualified or subscripted.
2. Qualification of a data-name can be provided through higher level data-names and file-names. Thus a data-name need not be unique within or between Record Descriptions provided a higher level data-name or a file-name can be used for qualification.
3. The key word FILLER may be used only to name an unreferenced elementary item in a record. Under no circumstances may a FILLER item be referred to directly.
4. A FILLER item must not have level-number 77.

Editing Clauses

FUNCTION: To permit suppression of non-significant zeroes and commas, to permit floating dollar signs or check protection, and to permit the blanking of an item when its value is zero.

$$[; \left\{ \begin{array}{l} \underline{\text{ZERO SUPPRESS}} \\ \underline{\text{CHECK PROTECT}} \\ \underline{\text{FLOAT DOLLAR SIGN}} \end{array} \right\} [\underline{\text{LEAVING}} \text{ integer } \left\{ \begin{array}{l} \text{PLACES} \\ \text{PLACE} \end{array} \right\}]] [\underline{\text{BLANK WHEN ZERO}}]$$

Notes:

1. The editing clauses can be specified only at the elementary item level.
2. The rules for editing, as shown in the MOVE verb, specify that data items are moved in conformity with the Record Description of the receiving item.
3. The three options, ZERO SUPPRESS, CHECK PROTECT, and FLOAT DOLLAR SIGN, all permit suppression of leading zeros and commas. If the LEAVING option is not employed, suppression will stop as soon as either a non-zero digit or the decimal point (actual or assumed) is encountered. Specifically:
 - a. When ZERO SUPPRESS is specified, leading zeros and commas will be replaced by spaces.
 - b. When CHECK PROTECT is specified, leading zeros and commas will be replaced by asterisks.
 - c. When FLOAT DOLLAR SIGN is specified, the rightmost character suppressed will be replaced by a dollar sign, and all other characters which are suppressed will be replaced by spaces.
4. The LEAVING option may be employed to stop suppression before the decimal point (actual or assumed) is encountered. When used, suppression stops (leaving "integer" positions to the left of the real or assumed decimal point) unless stopped sooner by the rules specified in Note 2. The "integer" position is a count of the number of characters, starting immediately at the left of the actual or assumed decimal point.
5. When the BLANK WHEN ZERO option is used, the item will contain nothing but spaces if the value of the item is zero. Thus, all other editing requirements, such as ZERO SUPPRESS, CHECK PROTECT, etc., will be overridden.

6. When any of these clauses are used, the item is assumed to be ALPHANUMERIC because the \$, *, and Δ are alphanumeric.
7. More comprehensive editing features are available in the PICTURE clause. When any of the above options are used the format of the item is assumed to contain editing symbols.
8. If both a PICTURE and editing clauses are present for an entry, the PICTURE takes precedence.

JUSTIFIED

FUNCTION: To specify non-standard positioning of a data item.

[; { JUSTIFIED } RIGHT]
 JUST

Notes:

1. The JUSTIFIED clause may appear only in elementary entries. It must not be specified for an item with any of the following properties:
 - a. CLASS NUMERIC (explicitly or implicitly);
 - b. USAGE other than DISPLAY; or
 - c. Actual or assumed decimal point specified.
2. When an item's storage area receives data as the result of an arithmetic or data movement procedure, the standard rules for positioning the value within the receiving area are as follows:
 - a. If the receiving item is NUMERIC, the value is aligned by decimal point, with zero fill (or truncation) on either end, as required.
 - b. If the receiving item is an edited item with an actual or implied decimal point, the value is aligned by decimal point, with fill or truncation on either end, as required.
 - c. If the receiving item is ALPHABETIC or ALPHANUMERIC (with no decimal point), the value is left-justified with space fill (or truncation) on the right.
3. JUSTIFIED RIGHT may be used to reverse the standard rule described in note 2.c., above. JUSTIFIED RIGHT causes the value to be right-justified with space fill (or truncation) on the left.
4. JUST is an abbreviation for JUSTIFIED.

Level Number

FUNCTION: To show the hierarchy of data within a logical record. To identify entries for condition-names, independent constants and working storage items, and RENAMES entries.

level-number

Notes:

1. A level-number is required as the first element in each Record Description entry.
2. A level-number may have values of 1-49, and 66, 77, and 88.
3. The level-number 1 signals the first entry in each Record Description. This corresponds to the logical record on which the READ and WRITE verbs operate.
4. Special level numbers have been assigned to certain entries where there is no real concept of level:
 - a. Level number 66 is assigned to identify RENAMES entries, and may only be used in Option 3 of the Record Description format.
 - b. Level number 77 is assigned to identify independent constants and independent working storage items.
 - c. Level number 88 is assigned to entries which define condition-names associated with a conditional variable, and may only be used with Option 4 of the Record Description format.

OCCURS

FUNCTION: To define tables of repeated items.

[; OCCURS [integer-1 TO]integer-2 TIMES [DEPENDING ON data-name-1]]

Notes:

1. This optional clause is used in defining tables of data items. References to table items require subscripts as described in Chapter 5.
2. This clause must not appear in entries with level number 1, 66, 77, or 88.
3. This clause is required when the data item either might not exist or might occur more than once. When OCCURS is not specified, one occurrence is assumed.
4. If integer-1 is not specified, integer-2 represents the exact number of occurrences. Integer-2 must not in any case be zero.
5. When the number of occurrences may vary, integer-1 specifies the minimum number of occurrences, and integer-2 specifies the maximum. Integer-1 may be zero to indicate that the data might not exist. Unless the DEPENDING option is also specified, the integer-1 option is regarded as documentation only. DEPENDING must not be specified unless integer-1 is specified.
6. The DEPENDING option is used to indicate that the number of occurrences is equal to the value of data-name-1. This value must be not less than integer-1 nor greater than integer-2. GE-600 Series COBOL requires data-name-1 to have USAGE COMPUTATIONAL-1; it must appear in the record to which the current Record Description entry pertains, and it must precede the variable portion of the record. Because of this rule, the data-name-1 entry must precede the OCCURS...DEPENDING entry itself.
7. If data-name-1 specifies fewer than integer-2 items, the significant items are understood to appear in successive positions at the beginning of the table; unused positions at the end of the table are called "table residue". Contents of the residue area are unpredictable.

8. The user should be aware that the results of OCCURS...DEPENDING generally differ from one machine line to another. In GE-600 Series COBOL, the results are as follows:
 - a. The DEPENDING option has no effect except in the FILE SECTION.
 - b. In the FILE SECTION, OCCURS...DEPENDING results in suppressing of table residue on the peripheral storage medium, as described below. This option also automatically causes the compiler to APPLY a PROCESS AREA to the file, whether or not the user specifies PROCESS AREA in the I-O-CONTROL paragraph.
 - c. When a WRITE references a record containing one or more OCCURS...DEPENDING items, the object-program examines the output record for opportunities for residue suppression. Such an opportunity is rejected unless two or more machine words can be suppressed. In the latter case, suppression proceeds on a machine word basis; the whole-word portion of the residue of each table is replaced by a single control word. If the residue of a table amounted to ten whole words, only one control word would appear instead in the output medium; the net saving would be nine words, in this case. Each variable length table in the record presents an opportunity for residue suppression. Actual suppression takes place in an implicit move from the PROCESS AREA to the output buffer.
 - d. When a READ references a file containing records described with OCCURS...DEPENDING, the object-program implicitly moves the record from the input buffer to the PROCESS AREA, expanding it to the format it had in memory prior to residue suppression.
 - e. When OCCURS...DEPENDING is used with any record of a file, there are several effects on the data format:
 - The Variable Length Record (VLR) format is automatically applied.
 - In addition to any residue suppression control words needed, each record in the peripheral medium begins with a control word required for reconstruction.
 - The RECORDING MODE must be BINARY (explicitly or implicitly).
9. Data-name-1 may be qualified, but it must not be subscripted.

10. A group item is said to have "variable length" if any item subordinate to it is described with the integer-1 option of OCCURS. Such an item is restricted in the following ways:
 - a. It cannot be subordinate to an OCCURS item.
 - b. It cannot be REDEFINED, or be subordinate to an item which is REDEFINED.
 - c. It cannot appear in a redefinition.
11. It is important to be aware that OCCURS does not imply SYNCHRONIZED. In the interest of object-program efficiency, consideration should be given to making table items explicitly SYNCHRONIZED.
12. A table item may be a conditional variable. The condition-name entries follow the conditional variable, as usual, and do not contain OCCURS clauses. Any references to such condition-names require subscripts.
13. The total size of the table is the product of the repeated item's size times the number of items in the table. The size given in a particular entry, however, refers to a single occurrence, not to the product of the size of a single occurrence times the number of occurrences.
14. If an item is described with this clause, its data-name must be subscripted in all references. If it is a group item, then each data-name belonging to the group must be subscripted whenever it is referenced. The data description clauses associated with an item whose description includes an OCCURS clause, apply to each repetition of the item being described.
15. Whenever possible, data items used as subscripts should have USAGE COMPUTATIONAL-1, to enhance object-program efficiency.

PICTURE

FUNCTION: To show a detailed picture of the Standard Data Format of an elementary item, the general characteristics of the item, and special report editing.

[; { PICTURE } IS character string]
PIC

Notes:

1. A PICTURE clause can be specified only at the elementary item level. (See Note 1 of the SIZE clause.)
2. PIC is an abbreviation for PICTURE.
3. A character string consists of any allowable combination of the characters and symbols described below.
4. The maximum number of characters or symbols allowed in a PICTURE character string is 30.
5. If the CLASS is given as ALPHANUMERIC in the same entry with a PICTURE, the CLASS remains ALPHANUMERIC regardless of the string of characters in the PICTURE. If the CLASS is given as ALPHABETIC or NUMERIC in the same entry with a PICTURE or at a higher level, the CLASS represented by the string of characters in the PICTURE must be the same as the explicit CLASS. If the CLASS given, implicitly or explicitly, is ALPHANUMERIC at a group level and is not stated in a subordinate level, the PICTURE, when used, determines the CLASS of the subordinate entry according to the rules given below.
6. The allowable characters (or symbols) which may appear in a character string are defined and described below according to the CLASS definition of the data item being described by the PICTURE clause and according to the associated MOVE category of the data item. (See CLASS clause and MOVE verb.)
 - a. NUMERIC characters (See CLASS, Note 3)
N-numeric (See MOVE, Note 2)
 - 9 indicates that the character position contains a NUMERIC character.
 - S indicates the presence of an operational sign (see SIGNED) which is not counted in the SIZE of the data item. The PICTURE of a CLASS IS NUMERIC data item can possess only one operational sign and if specified, "S" must be written as the left-most character of the PICTURE.

- V indicates an assumed decimal point which does not occupy a character position and is not counted in the SIZE of the data item. The PICTURE of a data item cannot contain more than one assumed decimal point.
- P indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character "P" is not counted in the SIZE of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) allowed in computations. (P cannot be used with COMPUTATIONAL [-n] items.) The scaling position character "P" can appear only to the left or right as a continuous string of "P" with a PICTURE description; since the scaling position character "P" implies an assumed decimal point (to the left of "P"s if "P"s are left-most PICTURE characters and to the right of "P"s if "P"s are right-most PICTURE characters), the assumed decimal point symbol "V" is considered redundant as either the left-most or right-most character within such a PICTURE description.
- O indicates the insertion character zero and is counted in the SIZE of the data item. If data is moved to a data item whose PICTURE description contains the insertion character "O", the numeric data is aligned by decimal point within the receiving character positions independently of the insertion characters. The insertion characters are placed in the receiving data item regardless of the nature of the sending data item. If data whose PICTURE description contains the insertion character "O" is moved to another data item, each data item character position is considered in the sending data item as part of the value for decimal point alignment purposes.
- b. ALPHABETIC characters (See CLASS, Note 4)
AB - alphabetic (See MOVE, Note 2)
- A indicates that the character position contains an ALPHABETIC character (letter or space).
- B indicates the insertion character space (blank) and is counted in the SIZE of the data item. If data is moved to a data item whose PICTURE description contains the insertion character "B", the alphabetic data is left justified within the receiving character positions independently of the insertion characters. The insertion characters are placed in the receiving data item regardless of the nature of the sending data item. If data whose PICTURE description contains the insertion character "B" is moved to another data item, each data item character position in the sending data item is considered as part of the data item during the move operation.

c. ALPHANUMERIC characters (See CLASS, Note 5)

1) AN - alphanumeric (See MOVE, Note 2)

X indicates the character position contains any allowable character in the computer's character set. If a data item PICTURE consists entirely of any combination of X, A and 9, other than all A's or all 9's, the PICTURE is considered and treated as if the entire PICTURE consist of all X's.

J,K J and K have special uses in some COBOL compilers. GE-COBOL accepts J and K as fully equivalent to X. However the understood initial VALUE of a WORKING-STORAGE item SPACES if the PICTURE contains J and no explicit VALUE clause is given. This feature simplifies conversion from certain other machine lines, and is not meant as a general substitute for VALUE SPACES.

2) AE - Alphanumeric edited (See MOVE, Note 2)

A PICTURE description containing at least one of the insertion characters "B" or "0" and at least one of the character "X" or a PICTURE description containing at least one of the insertion character "0" and at least one of the characters "X" or "A" is considered an alphanumeric edited data item.

3) NE - numeric edited (See MOVE, Note 2)

Numeric edited implies:

- a) the data item being described is ALPHANUMERIC.
- b) the data item being described is a numeric edited data item.
- c) the data item can only receive data which is numeric in content.
- d) the maximum number of digit positions that may be represented is 18.

9, V, and P have been defined under NUMERIC characters above. 0 and B are repeated in definition here to indicate special editing cause and effect.

indicate the insertion characters comma, space (blank) and zero respectively; each insertion character is counted in the SIZE of the data item but does not represent a numeric character position. The presence of zero suppression or replacement of zeroes by spaces (Z) and check protection (*) indicate that suppression of leading insertion characters also takes place with associated space or asterisk replacement.

The floating characters, floating dollar (\$\$), floating plus (++) , and floating minus (--), "float through" the insertion characters, i.e., if the most

significant digit is immediately to the right of an insertion character, the floating replacement character replaces the insertion character.

- . indicates the actual decimal point and is a special insertion character. The data item being edited is aligned by decimal point and the actual decimal point appears in the indicated character position. The actual decimal point, unlike the assumed decimal point (V), is counted in the SIZE of the data item. A data item cannot contain more than one assumed or actual decimal point, i.e., the symbols "V" and "." are mutually exclusive within a PICTURE description. A PICTURE description must not terminate with the symbol ".", unless immediately followed by one of the punctuation characters, semicolon, or period.
- + } indicates the editing sign control characters. As
 - } fixed insertion characters, the "+" and "-" symbols
 CR } can be used only at the beginning or at the end of a
 DB } PICTURE; "CR" and "DB" can be used only at the extreme
 right end of a PICTURE and represent two character
 positions when counted in the SIZE of the data item.
 When the fixed insertion characters "+" or "-" are
 used at the beginning, they must be the left-most
 character in the PICTURE. As floating characters,
 the "+" and "-" characters are written from the
 extreme left to represent each leading numeric charac-
 ter position into which the editing sign may be
 floated. A single editing sign will be placed in the
 least significant position shown by the character "+"
 or "-" in the PICTURE description (including the in-
 sertion characters ",", "B" or "0") immediately pre-
 ceding the first non-zero digit in the data or the
 decimal point "V" or ".", whichever is encountered
 first. One exception is that if all data character
 positions in the PICTURE contain the floating charac-
 ters "+" or "-", then the entire data item will con-
 sist of spaces when the value of the data item is zero.
 A floating "+" or "-" may appear to the right of a
 decimal point in a PICTURE only if all character
 positions are represented by "+" or "-"s. The data
 item must contain at least one more editing sign
 character position than the maximum number of signi-
 ficant digits in the associated source data item.
 Editing signs are counted in the SIZE of the data item.
 Depending on the value of the data item associated with
 the PICTURE, the display character signs outlined in the
 following table are produced. "+", "-", "CR" and "DB"
 are all mutually exclusive.

PICTURE Sign

Display Representation
Data Item Positive-Data Item Negati

+
-
CR
DB

+
Space
Spaces (2)
Spaces (2)

-
-
CR
DB

- Z indicates the standard zero suppression and replacement of zeroes by spaces character. The character "Z" represents each leading numeric character position which is to be replaced by a space for each unwanted left-hand zero. Replacement of zeroes by spaces terminates with the character (including the insertion characters ",", "B" and "0") immediately preceding the first non-zero digit in the data, or the decimal point "V" or ".", whichever is encountered first. One exception is that if all data character positions in the PICTURE contain the replacement character "Z", then the entire data item will consist of spaces when the value of the data item is zero. A "Z" may appear to the right of a decimal point in a PICTURE only if all numeric character positions are represented by "Z"s. The replacement of zeroes by spaces (character "Z") is counted in the SIZE of the data item.
- * indicates the asterisk replacement character and designates each leading numeric character position which is to be replaced by an asterisk for each unwanted left-hand zero. Asterisk replacement terminates with the character (including the insertion characters ",", "B", and "0") immediately preceding the first non-zero digit or the decimal point "V" or ".", whichever is encountered first. One exception is that if all numeric character positions in the PICTURE contain the replacement character asterisk (*) then the entire data item will consist of asterisks when the value of the data item is zero, except that the decimal point (.) will be retained and printed.

An "*" may appear to the right of a decimal point in a PICTURE only if all numeric character positions are represented by "*"s. The asterisk is counted in the SIZE of the data item.

- \$ indicates the dollar sign character. As a fixed insertion character, the dollar sign may appear only once in a PICTURE. As a floating character, the dollar sign is written from the extreme left to represent each leading numeric character into which the dollar sign may be floated. A single dollar sign will be placed in the least significant position shown by the character "\$" in the PICTURE (including the insertion characters ",", "B", and "0") immediately preceding the first non-zero digit in the data, or the decimal point "V" or ".", whichever is encountered first. One exception is that if all the numeric character positions in the PICTURE contain the floating character "\$", then the entire data item will consist of spaces when the value of the data item is zero. The data item must contain at least one more dollar sign character position than the maximum

number of significant digits in the associated source data item. A "\$" may appear to the right of a decimal point in a PICTURE only if all numeric positions are represented by "\$"s. The dollar sign is counted in the SIZE of a data item.

4) FE - floating point numeric edited

A special editing option is provided for items with USAGE DISPLAY-1 explicitly specified. Such an item is considered an edited floating point item. This option is not a standard COBOL feature. The PICTURE of a DISPLAY-1 item must conform to the following format:

$$\left\{ \begin{array}{c} + \\ - \end{array} \right\} 9.9(n)E \left\{ \begin{array}{c} + \\ - \end{array} \right\} 99$$

The first character must be + or -, the report sign for the mantissa. The next characters represent the desired number of digits in the mantissa; n must be a one or two-digit integer from 1 to 17. The E is required; and represents an E insertion character which is counted in the item's size. The remaining characters must be + or - (the report sign for the exponent) and two 9's representing the exponent itself. Leading and/or trailing B's may be appended to the above format if needed.

7. Miscellaneous Notes

- a) A PICTURE must consist of at least one of the following characters:

A X 9 * Z J K

or at least a pair of one of the following characters:

+ - \$ (indicating floating characters).

- b) Only one type of floating replacement character, i.e., "\$", "+", "-", "*", or "Z" can be used within a given PICTURE description. The replacement characters "*" or "Z" may be preceded by a fixed "\$"; "\$" (fixed or floating), "Z" or "*" may be used with a fixed "+" or "0". A PICTURE character "9" can never appear to the left of a floating or a replacement character.
- c) An integer which is enclosed in parentheses following the symbols "A", ",", "X", "9", "P", "Z", "*", "B", "\$", "0", "+", or "-" indicates the number of consecutive occurrences of the symbol. (e.g., P(10)9(2) and P(10)99 and P(10)999 are all equivalent.) Note that the following symbols may appear only once in a given PICTURE, "S", "V", ".", "CR" and "DB".

- d) If editing clauses are used in conjunction with a PICTURE clause, the two sets of clauses must not be contradictory. The editing clause, CHECK PROTECTION, and the PICTURE character "*" are mutually exclusive with the editing clause BLANK WHEN ZERO.
- e) Editing takes place in the object program in the more common usages when any of the following events occur:
- A MOVE statement with an edited numeric receiving item and an elementary numeric sending item is executed.
 - An arithmetic verb statement with an edited numeric receiving item is executed.
 - A report group containing an edited numeric receiving item is presented. (The item's description may specify either a SOURCE clause, entailing a numeric elementary source item, or a SUM clause.)

POINT LOCATION

FUNCTION: To define an assumed decimal point or binary point.

[; POINT LOCATION IS { LEFT / RIGHT } integer { PLACES / BITS }]

Notes:

1. This optional clause may appear only in elementary entries.
2. The decimal point location is normally defined via PICTURE, but an assumed decimal point may be specified via the PLACES option of the POINT clause. If both are specified, they must agree.
3. The POINT clause indicates the position of an assumed decimal point only, never an actual decimal point. Actual decimal points can only be specified via PICTURE.
4. When the PLACES option is used, the assumed decimal point is integer decimal places to the LEFT or RIGHT of the least-significant position of the item. PLACES is implied when neither PLACES nor BITS is specified.
5. When the BITS option is used it has no effect, but may be used for documentation only.
6. If USAGE is COMPUTATIONAL[-n], the point location must not be RIGHT and integer must not exceed the item's size.

RANGE

FUNCTION: To specify the potential range of the value of an item.

[; RANGE IS literal-1 THRU literal-2]

Notes:

1. This optional clause may be used for documentation.
2. For NUMERIC items literal-1 and literal-2 represent the respective minimum and maximum values of the item. Literal-2 must not contain more digits than are specified in the SIZE clause.
3. For non-NUMERIC items, each character of literal-1 and literal-2 represents the respective minimum and maximum values of the corresponding character position in the item.
4. RANGE can be written only at the elementary item level.
5. Literal-1 and literal-2 may be Figurative Constants.

REDEFINES

FUNCTION: To allow the same computer storage area to contain different data items.

level-number data-name-1 [; REDEFINES data-name-2]

Notes:

1. The **REDEFINES** clause, when used, must immediately follow data-name-1.
2. The level-numbers of data-name-1 and data-name-2 must be identical.
3. Redefinition starts at data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered.
4. When the level-number of data-name-2 is other than 1, it must specify a storage area of the same size as data-name-1.
5. The entries giving the new description of the storage must immediately follow the entries describing the area being re-defined. Thus multiple redefinitions of an area must be "chained", rather than all of them being related to the original definition. A special consideration for redefinition of **SYNCHRONIZED** items is discussed in the notes under the **SYNCHRONIZED** clause.
6. This clause must not be used in level 1 entries in the **FILE SECTION**. Implicit redefinition is provided by the **DATA RECORDS** clause in the File Description entry.
7. **Data-name-2** never needs to be qualified.
8. The Record Description entry for data-name-2 may not contain an **OCCURS** clause, nor may data-name-2 be subordinate to an entry which contains an **OCCURS** clause.
9. The entries giving the new description of the storage area must not contain any **VALUE** clauses, except in condition-name entries.

RENAMES

FUNCTION: To permit alternative, possibly overlapping, groupings of elementary items.

66 data-name-1 RENAMES data-name-2 [THRU data-name-3]

Notes:

1. One or more RENAMES entries can be written for items or groups within a logical record.
2. All RENAMES entries associated with a given logical record must immediately follow its last record description entry.
3. Data-name-2 and data-name-3 must be names of elementary items or groups of elementary items in the associated logical record, and cannot be the same data-name. A 66 level entry cannot rename another 66 level entry nor can it rename a 77, 88, or 01 entry.
4. When data-name-3 is specified, data-name-1 is a group item which includes all elementary items starting with data-name-2 (if data-name-2 is an elementary item) or the first elementary item in data-name-2, and concluding with data-name-3 (if data-name-3 is an elementary item) or the last elementary item in data-name-3 (if data-name-3 is a group item).
5. When data-name-3 is not specified, data-name-2 can be either a group or an elementary item; when data-name-2 is a group item, data-name-1 is treated as a group item, and when data-name-2 is an elementary item, data-name-1 is treated as an elementary item.
6. Data-name-2 must precede data-name-3 in the record description.
7. Data-name-3 cannot be contained within data-name-2.
8. Data-name-1 cannot be used as a qualifier, and can be qualified only by the names of the level 1 or FD entries. Neither data-name-2 nor data-name-3 may have an OCCURS clause in its record description entry nor be subordinate to an item that has an OCCURS clause in its record description entry.
9. Data-names must be unique.

SIGN

FUNCTION: To specify the presence of a standard operational sign in an elementary item.

[; SIGNED]

Notes:

1. An item whose description specifies an operational sign must be NUMERIC. Therefore, its CLASS need not be specified.
2. SIGNED indicates the use of a standard operational sign. (This may also be indicated by the use of an "S" in the PICTURE.) A standard operational sign is not considered in determining the SIZE.
3. An item which contains any editing symbols other than 0 cannot have an operational sign.
4. This clause can be used only at the elementary item level.

SIZE

FUNCTION: To specify the size of an item, in terms of the number of standard data format characters.

[; **SIZE** IS integer { CHARACTER [S]
DIGIT [S] }]

Notes:

1. The size of an item must be specified at the elementary item level by means of a **SIZE** clause or a **PICTURE**. A **PICTURE** and a **SIZE** clause need not both be given. If both are given, they must agree. Use of the **SIZE** clause at any level other than the elementary item level is optional. If **SIZE** is specified at a group level, the **SIZE** of the group is the sum of the sizes (as determined from **SIZE** or **PICTURE**) of the elementary items comprising the group.
2. Integer represents the exact number of characters excluding operational symbols.
3. Any of the key words of the **USAGE** and/or **CLASS** clauses can be inserted between integer and the word **CHARACTERS** (or **DIGITS**) in the **SIZE** clause format. If this is done, separate **USAGE** and/or **CLASS** clauses must not be written. For example, note the following:

02 PAGE; SIZE IS 7 NUMERIC COMPUTATIONAL DIGITS; VALUE IS
0000342.

SYNCHRONIZED

FUNCTION: To specify positioning of an elementary item with a computer word or words.

[; { SYNCHRONIZED } { LEFT } { RIGHT }]
 { SYNC }

Notes:

1. This clause indicates that the COBOL Processor, in creating the Internal Format of this item, must place the item in the minimum number of computer words which can contain the item, with no part of any other item sharing those words.
2. The computer word, or words, containing the SYNCHRONIZED item may also have to contain some unused character positions in order to fill the computer word, or words. When SYNCHRONIZED LEFT is specified, these unused character positions (if any) will occupy the least significant portion of the (last) word. When SYNCHRONIZED RIGHT is specified, the unused positions (if any) will occupy the most significant portion of the (first) word. It is improper to attempt to describe the unused character positions with FILLER items.
3. All unused character positions resulting from the SYNCHRONIZED clause appear in the external format.
4. Whenever a SYNCHRONIZED item is referenced in the source program, the original size of the item, as shown in the SIZE or PICTURE clause, and excluding any embedded unused character positions, is used in determining any action which depends on size, such as justification or truncation. It is important to observe, however, that the REDEFINES clause leads to a redefinition of a memory area, not simply of the data items occupying the area. If SYNCHRONIZED clauses resulted in unused character positions in the original definition of the area, other than to the left of the first item being redefined, the new definition must account for all such character positions. If the first item in the original definition is SYNCHRONIZED RIGHT, the area being redefined begins in the leftmost character of the first word allocated to the original item. If the last item of the original definition is SYNCHRONIZED LEFT, the area being redefined extends to the rightmost character of the last word allocated to the original item.
5. When SYNCHRONIZED is specified for an item within the scope of an OCCURS clause, each occurrence of the item will be SYNCHRONIZED.
6. Items whose USAGES are COMPUTATIONAL are automatically SYNCHRONIZED.
7. The SYNCHRONIZED clause may be used only for elementary items.
8. SYNC is an abbreviation for SYNCHRONIZED.

USAGE

FUNCTION: To specify the dominant use of a data item.

[; USAGE IS {
COMPUTATIONAL
COMP
COMPUTATIONAL-1
COMP-1
COMPUTATIONAL-2
COMP-2
COMPUTATIONAL-3
COMP-3
DISPLAY
DISPLAY-1
DISPLAY-2
}

Notes:

1. This optional clause may be written at any level. If the USAGE of an item is not specified, it is assumed to be DISPLAY.
2. COMP is an abbreviation for COMPUTATIONAL.
COMP-n is an abbreviation for COMPUTATIONAL-n.
3. The optional suffixes on the words COMPUTATIONAL (-1,-2,-3) and DISPLAY (-1,-2) are used when special internal data formats are required. The non-suffixed usages are preferred. If USAGE is specified at a group level, it applies to all of the subordinate elementary items, and they must not have contradicting usages (this includes being differently suffixed). When no special distinction is made, any discussion in this manual of COMPUTATIONAL or DISPLAY items applies equally to suffixed and non-suffixed usages.
4. A COMPUTATIONAL (or COMPUTATIONAL-n) is stored as a SYNCHRONIZED binary number and must be NUMERIC. Its CLASS therefore need not be specified, and its description must conform to the rules for NUMERIC items. A group item described as COMPUTATIONAL or COMPUTATIONAL-n cannot itself be used in computations; instead, the specified USAGE applies individually to each subordinate elementary item. The COMPUTATIONAL-3 USAGE is intended for use only to achieve compatibility with non-COBOL programs.
5. An item's external format (as it is stored in a peripheral medium) and its internal format (as it is stored in the computer memory) are always the same. However, the USAGE clause permits a choice of several optional formats, and the item's description must conform to the rules of the pertinent format.

COMPUTATIONAL signifies decimal precision binary. If the item's description places it into one of the categories in the following table, it is stored as a single length number; otherwise it is double length.

<u>Single length decimal precision numbers</u>	
<u>No. of integral digits</u>	<u>No. of fractional digits</u>
1-8	0
0-5	1-3
0-3	4-5
0	6-8

COMPUTATIONAL-1 signifies binary integer. The item must have only integral digits. If it has 1-8 digits, it is stored as a single length number; if it has 9-18 digits, it is double length.

COMPUTATIONAL-2 signifies floating point binary. If the mantissa has 1-8 digits of decimal significance, it is stored as a single length number; if it has 9-18 digits, it is double length.

COMPUTATIONAL-3 signifies single length fixed point binary integer. The item may have at most 10 integral digits.

DISPLAY signifies standard data format.

DISPLAY-1 signifies edited floating point. The item's CLASS is implicitly ALPHANUMERIC.

DISPLAY-2 signifies Commercial Collating Sequence. Each character of a DISPLAY-2 item is represented in the computer system in a special way, so that the result of a comparison of two DISPLAY-2 items conforms to the Commercial Collating Sequence rather than to the machine's standard collating sequence. A DISPLAY-2 item must have ALPHANUMERIC or ALPHABETIC CLASS. It is against COBOL rules to compare a DISPLAY-n item with any item having a different USAGE. DISPLAY-2 items are sensible only within the computer system; such an item must be explicitly moved to/from a DISPLAY item if it is to appear on the punched cards, a printer listing, or similar external media. This function cannot be accomplished by REDEFINES.

VALUE

FUNCTION: To define the value of constants, the initial value of working-storage items, or the values associated with a condition-name.

Option 1:

[; VALUE IS literal]

Option 2:

[; { VALUE IS
VALUES ARE } literal-1 [THRU literal-2] [,literal-3
[THRU literal-4 ...]]

Notes:

1. If the entry is not a condition-name entry, the interpretation of the VALUE clause depends upon the Data Division Section in which the entry is described.
 - a. The VALUE clause has no effect in the FILE SECTION, but may be used for documentation only.
 - b. For Working Storage and Constant Storages the item will contain the specified VALUE at the start of the object-program.
2. Option 2 can be used only in connection with condition-names. When the THRU option is used, literal-1 must be less than literal-2, literal-3 must be less than literal-4, etc.
3. A figurative constant may be substituted in the format above wherever a literal is specified.
4. The VALUE clause must not be stated in a Record Description entry which contains an OCCURS clause, or in an entry which is subordinate to an entry containing an OCCURS clause. This rule does not apply to condition-name entries. A similar rule applies to the context of a REDEFINES clause.
5. If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a non-numeric literal, and the group area will be initialized without consideration for the individual elementary or group items contained within this group. VALUE cannot be stated at the subordinate levels within the group, nor can VALUE be specified for a group containing items requiring separate handling due to synchronization, USAGE, etc.

6. The VALUE clause must not conflict with other clauses in the Data Description of the item or in the Data Description within the hierarchy of the item. The following rules apply:
 - a. If the item is NUMERIC (explicitly or implicitly), all literals in the VALUE clause must be numeric literals; if the item is ALPHABETIC or ALPHANUMERIC, all literals must be non-numeric.
 - b. All non-numeric literals in a VALUE clause must have no more characters than the SIZE (explicitly or implicitly) in the Data Description of the item indicates.
 - c. All numeric literals in a VALUE clause of an item must have a value which is within the range of values indicated by the SIZE, RANGE, or PICTURE clauses, e.g., for PICTURE PPP99 the literal must be within the range .00000-.00099.
7. When VALUE is not specified, the initial contents of the working storage areas may be unpredictable.
8. The VALUE clause cannot be used for items whose USAGE is DISPLAY-2. DISPLAY-2 items cannot be conditional variables.

Specific Entry for a Condition-name

Each condition-name requires a separate entry with level number 88. This entry contains the name of the condition and the value, values, or range of values associated with the condition-name. The condition-name entries for a particular conditional variable must follow the entry describing the item with which the condition-name is associated. A condition-name can be associated with any elementary or group item, except the following:

1. Another condition-name.
2. A level 66 item.
3. A group containing items requiring separate handling due to synchronization, usage, and so forth.

More specifically, some of the possible ways of writing condition-names entries are:

nn data-name

88 condition-name-1 VALUE IS literal-1.

88 condition-name-2 VALUES ARE literal-2, literal-3.

88 condition-name-3 VALUES ARE literal-4 AND literal-5 AND...,

88 condition-name-4 VALUES ARE literal-6 THRU literal-7.

As an example:

03 GRADE SIZE IS 2 CHARACTERS.

88 PRIMARY VALUE IS 1.

88 SECOND VALUE IS 2.

.

.

.

88 GRADE-SCHOOL VALUES ARE 1 THRU 6.

88 JUNIOR-HIGH VALUES ARE 7 THRU 9.

88 HIGH-SCHOOL VALUES ARE 10 THRU 12.

88 GRADE-ERROR VALUES ARE 0, AND 13 THRU 99.

REPORT WRITER

GENERAL DESCRIPTION

A report represents a pictorial organization of data. To present a report, the physical aspects of the report format must be differentiated from the conceptual characteristics of the data to be included in the report. In defining the physical aspects of the report format, consideration must be given to the width and length of the report medium, to individual page structure, to the type of hardware device on which the report is finally written. Structure controls are established to insure the report format is maintained.

To define the conceptual characteristics of the data, i.e., the logical organization of the report itself, the concept of level structure is used. Each report may be divided into respective report groups, which, in turn, are subdivided into a sequence of items. Level structure permits the programmer to refer to an entire report-name, a major report-group, a minor report-group, an elementary item within a report-group, etc.

To create the report, the approach taken is to define the types of report groups that must be considered in presenting data in a formal manner. Types may be defined as heading groups, footing groups, control groups or detail print groups. A report group describes a set of data that is to be considered as an individual unit, irrespective of its physical format structure. The unit may be the presentation of a data record, a set of constant report headings, or a series of variable control totals. The description of the report group is a separate entity. The report group may extend over several actual lines of a page and may be of any type described above which is necessary to produce the desired output report format.

Report Section

The Report Section contains two elements: the description of the overall structure of the report and the description of the report-group, including the description of the elementary items within each report-group.

Each description consists of a level indicator, a data-name, if required, and a series of independent clauses which are terminated by a period. The contents of the descriptions depend on certain clauses which are optional and therefore dependent on the requirements of the user.

Report Name Description Entry

This entry contains the information pertaining to the overall format of the report and is indicated by the level indicator RD. The characteristics of the report page are outlined and provided with limits by describing the number of physical lines per page and the limits for presenting specified

headings, footings, details within a page structure. Data items which act as control factors during presentation of the report are specified in the RD entry. Each report associated with an output file description must be defined by an RD entry.

Report Group Description Entry

A report group may be a set of data made up of several print lines with many data items or the report group may consist of one print line with one data item. Report-groups may exist within report groups -- all or each capable of reference by a GENERATE or USE statement in the Procedure Division. A description of a set of data becomes a report group by the presence of a level number and a TYPE description. The level number gives the depth of the group and the TYPE describes the purpose of the report group presentation. If report groups exist within report groups, they must all have the same TYPE description. Including a data-name with this entry permits the group to be referred to by a GENERATE or a USE statement in the Procedure Division. At object program time, report groups are created as a result of Report Writer GENERATE statements in the Procedure Division.

This entry defines the characteristics for a report group, whether a series of lines, a line, or an elementary item. The placement of an item in relation to the entire report group, the hierarchy of a particular report group within a report group, the format description of all items and any control factors associated with the group are all defined in the entry. The system of level numbers is employed here to indicate elementary items and group items of data within the range 01 to 49.

Pictorially to the programmer, a report group is a line, or a series of lines, initially consisting of all SPACES; its length is determined by the compiler based on environmental specifications. Within the framework of a report, the order of report groups specified is not significant. Within the framework of the report group, the programmer describes the presented elements consecutively from left to right and then from top to bottom. The description of a report group, analogous to the Data Record, consists of a set of entries defining the characteristics of the included elements. However, in the report group, SPACES are assumed except where a specific entry is indicated for presentation, whereas in the data record, every character position must be defined.

DEFINITIONS

The following terms are defined according to their meaning in the Report Writer language.

REPORT	a formal presentation of a printed set of data.
PAGE	a vertical division of a report representing a physical separation of continuous report data; the separation may be based on internal reporting requirements and/or external characteristics of the reporting medium.

FORMAT	a specific arrangement of a set of data within the structure of a page and/or report.
LINE	a horizontal division of a page representing one row of characters.
COLUMN	a specific position within the line.
REPORT-GROUP	an integral set of related data within a report.
PRINT GROUP	equivalent to a report group in the Report Section.
DATA-NAME	a name designating a group of data or an elementary item of data in the Data Division.
GROUP-ITEM	an item within a report group containing subordinate items; the report group itself is a group item.
ELEMENTARY ITEM	an item within a report group containing no subordinate items.
CONTROL HIERARCHY	a designated order of specific elements of information: within the order, any change in the value of the designated elements produces a control break.
CONTROL BREAK	a term describing the recognition of a change in the value of a data item designated as an element in the CONTROL hierarchy. The change is noted as a difference in the value of a source data item between the execution of the previous GENERATE and the execution of the present GENERATE. A control break sets in motion the automatic report writer function of producing special CONTROL footing and/or CONTROL heading report groups associated with the data item in which the change occurred.
CONTROL GROUP	an integral set of related data in a report specifically associated with a data item in the CONTROL hierarchy. The entire set of CONTROL HEADING report-group(s), CONTROL FOOTING report-group(s) and associated DETAIL report-group(s) comprise the control group for a given CONTROL data-name. Within the CONTROL hierarchy, lower level CONTROL HEADING report group(s) and CONTROL FOOTING report group(s) are implicitly included in a higher level control group.

COUNTER a device into which numerical units of information can be added or subtracted dependent on the arithmetic operation and/or operational sign of the information.

PAGE BREAK the event of advancing to a new page during the presentation of a report.

LINE-COUNTER

LINE-COUNTER is the fixed data-name for a counter automatically supplied by the Report Writer.

LINE-COUNTER is used by the Report Writer to automatically generate PAGE/OVERFLOW HEADING and/or PAGE/OVERFLOW FOOTING report groups.

LINE-COUNTER may be referred to by Procedure Division statements. Since more than one Report Name Description Entry may exist in the Report Section, the user must qualify LINE-COUNTER by the data-name of the report in the Procedure Division when necessary.

Since the LINE-COUNTER is used in conjunction with the PAGE LIMIT(S) clause for test and control purposes within the Report Writer, the user must be careful not to change the LINE-COUNTER by Procedure Division statements. Changing the LINE-COUNTER may cause PAGE format control to become unpredictable in the Report Writer.

LINE-COUNTER is automatically set, reset, tested, incremented, etc., by the Report Writer based on control specifications in the PAGE LIMIT(S) clause and values specified in the LINE NUMBER and NEXT GROUP clauses.

LINE-COUNTER is automatically set to zero initially by the Report Writer; likewise, LINE-COUNTER is automatically reset to zero whenever a page break occurs.

If a relative LINE NUMBER or relative NEXT GROUP indication exceeds a PAGE LIMIT specification during object time, a page break occurs.

The value of LINE-COUNTER during any Procedure Division test statement represents the last LINE NUMBER printed on from the previous generated report group or represents the last LINE NUMBER skipped from a previous NEXT GROUP specification.

PAGE-COUNTER

PAGE-COUNTER is automatically generated by the Report Writer to be used as a SOURCE data item in order to present consecutive page numbers within a report group.

PAGE-COUNTER is the fixed data-name for a counter automatically supplied by the Report Writer if PAGE-COUNTER is given as a SOURCE data name in a report group description entry. One PAGE-COUNTER is supplied for each report described in the Report Section.

If PAGE-COUNTER is given as the SOURCE of more than one data item, the number of NUMERIC characters indicated by the SIZE or PICTURE clause must be identical. The PICTURE clause must indicate sufficient NUMERIC character positions to prevent overflow.

PAGE-COUNTER may be referred to by Procedure Division statements. Since more than one Report Name Description Entry may exist in the Report Section, the user must qualify PAGE-COUNTER by the data-name of the report in the Procedure Division when necessary.

PAGE-COUNTER is initially one (1). If a starting value for PAGE-COUNTER other than one desired, the programmer may reset PAGE-COUNTER by a Procedure Division statement.

PAGE-COUNTER is automatically incremented by one (1) each time a page break is recognized by the Report Writer, after the production of any PAGE or OVERFLOW FOOTING report group but before production of any PAGE or OVERFLOW HEADING report group.

ENTRY FORMATS

General Notes

Report Section entries consist of a level indicator and a series of independent clauses. The mnemonic level indicator RD is used to identify the start of a Report Name Description Entry and distinguishes this entry from those describing a Report Group Description Entry.

Specific Formats

The Report Writer clause formats are arranged in an alphabetic order in this section for both the Complete Entry Skeleton and the individual clause function descriptions.

RD Entry

Complete Entry Skeleton

FUNCTION: To furnish information concerning the physical structure and overall characteristics of a report.

Option 1:

RD data-name-1 [WITH CODE mnemonic-name] COPY library name.

Option 2:

RD data-name-1 [WITH CODE mnemonic-name] [; CONTROL[S] ..]

[; PAGE LIMIT[S] ..] .

Notes:

1. The RD level indicator identifies the beginning of a Report Name Description and precedes the data-name. Data-name-1 must be unique.
2. The RD entry is terminated by a period.

CODE

FUNCTION: To affix a unique character, identifying this report, to each report-group GENERATED in the report.

[WITH CODE mnemonic-name]

Notes:

1. Mnemonic-name must be identified with a unique character in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION. (It is the user's responsibility to assure unique character assignment if several reports are produced in the program.)
2. As the RD Complete Entry format shows, the CODE clause, when used, must follow immediately after the report name (data-name-1).

CONTROL(S)

FUNCTION: To indicate the data-name(s) which specify the control hierarchy for this report.

[; { CONTROL IS
CONTROLS ARE } [FINAL] [,data-name-1] [,data-name-2..] [,data-name-n]

Notes:

1. The CONTROL(S) clause is required when CONTROL HEADING and/or CONTROL FOOTING report groups are specified.
2. The data-names indicating the control break items are listed in order from major to minor; FINAL is the highest CONTROL, data-name-1 is the major CONTROL, data-name-2 is the intermediate CONTROL, data-name-n is the minor CONTROL.
3. FINAL is a fixed data-name indicating the highest CONTROL for this report. CONTROL FOOTING TYPE report groups which require presenting only once on conclusion of the report are associated with the CONTROL hierarchy FINAL and are produced when TERMINATE is executed.
4. The data-name(s) specified in the CONTROL(S) clause are referred to by the TYPE clause in the Report Group Description Entry.
5. The data-names other than FINAL must be defined in the File or Working-Storage Section of the Data Division.

COPY

FUNCTION: To obtain a data-name description entry from the COBOL library.

RD data-name-1 [WITH CODE mnemonic-name] COPY library-name.

Notes:

1. The data-name replaces the library data-name in the library-name entry when the COPY clause is compiled; all of the descriptive clauses in the library entry are included as part of the data-name-1 description entry except the CODE clause.

PAGE LIMIT(S)

FUNCTION: To indicate the specific line control to be maintained within the presentation of a PAGE.

[; PAGE { LIMIT IS
LIMITS ARE } integer-1 { LINE
LINES }
[, HEADING integer-2] [, FIRST DETAIL integer-3]
[, LAST DETAIL integer-4] [, FOOTING integer-5]]

Notes:

1. The PAGE LIMIT(S) clause is required when PAGE format must be controlled by the Report Writer. The PAGE LIMIT(S) clause may be omitted when no association is desired between report group(s) and the physical format of an output PAGE, e.g., a report consisting solely of TYPE DETAIL report groups which are printed on the final reporting medium continuously.
2. PAGE LIMIT integer-1 LINES is required to specify the depth of the report PAGE; the depth of the report PAGE may or may not be equal to the physical perforated continuous form often associated in a report with the page length.
3. Integer-2 through integer-5 each must either be less than or equal to integer-1.
4. If absolute line spacing is indicated for all the report group(s), none of the integer-2 through integer-5 controls need to be specified.
5. If relative spacing is indicated for individual report group entries, one or more of the above LIMIT(S) must be defined, dependent on TYPE of report group(s) within the report, in order for the Report Writer to maintain control of PAGE format.
6. The PAGE LIMITS options are defined as follows:

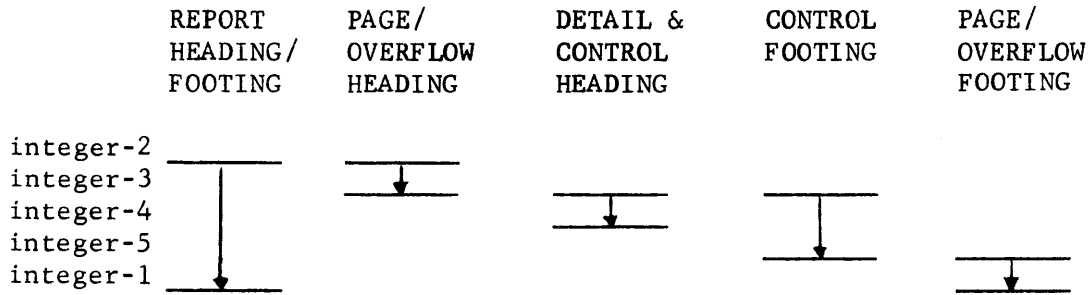
HEADING integer-2: the first line number of the first HEADING print group. No print group will start preceding integer-2.

FIRST DETAIL integer-3: the first line number of the first normal print group, i.e., body; no DETAIL or CONTROL print group will start before integer-3.

LAST DETAIL integer-4: the last line number of the last normal print group, i.e., body; no DETAIL or CONTROL HEADING will extend beyond integer-4.

FOOTING integer-5: the last line number of the last CONTROL FOOTING print group; no CONTROL FOOTING print group will start before integer-3 nor extend beyond integer-5. TYPE PAGE FOOTING or TYPE OVERFLOW FOOTING print groups will follow integer-5.

7. If neither integer-4 nor integer-5 is specified, both are considered to be equivalent to integer-1 value.
8. The following chart pictorially represents PAGE format as specified by all PAGE LIMIT(S) options:



9. Absolute LINE NUMBER or absolute NEXT GROUP spacing, must be consistent with controls specified in the PAGE LIMIT(S) clause.
10. Only one PAGE LIMIT(S) clause may be specified per Report Name Description Entry.

Report Group Entries

Complete Entry Skeleton

FUNCTION: To specify the characteristics of a particular report group and the individual items within a report group.

Option 1:

nn [data-name-1] COPY data-name-2 [FROM LIBRARY].

Option 2:

nn [data-name-1] [; CLASS...] [; LINE NUMBER...] [; NEXT GROUP...]
[; SIZE...] [; TYPE... .]

Option 3:

nn [data-name-1] [; CLASS...] [; COLUMN NUMBER..] [; editing clauses]
[; GROUP INDICATE] [; JUSTIFIED...] [; LINE NUMBER...]
[; PICTURE...] [; POINT LOCATION] [; SIZE...] [RESET...]
{ ; SOURCE...
; SUM...
; VALUE... } [; USAGE...]

Notes:

1. Option 2 is used to indicate a report group; the report group extends from this entry to an equal or higher level entry.
2. Option 3 is used to indicate an elementary or group item within a report group. If a report group consists of only one elementary entry, Option 3 may include the TYPE and NEXT GROUP clause in order to specify the report group and elementary item in the same entry.
3. In Option 2 or 3, to refer to the report group by a GENERATE or USE statement in the Procedure Division requires the presence of a data-name-1, together with the TYPE and level number.

4. If LINE NUMBER is not given in the level 01 entry, it must be given at a subordinate group level prior to the first elementary item within the report group. If LINE NUMBER is specified for a report group, the entire report group is presented on the specified LINE NUMBER. If LINE NUMBER is specified for a group item, it applies to all subordinate items, and no lower level LINE NUMBER clause is permitted. The following general rules apply for use of the LINE NUMBER and NEXT GROUP clauses:
 - a. A line definition must start with a LINE NUMBER clause whether any pre-slewing is desired or not. "LINE NUMBER IS PLUS integer" actually results in integer minus 1 lines of pre-slewing. Therefore, "LINE NUMBER IS PLUS 1" results in no pre-slewing. "LINE NUMBER IS PLUS 0" may be specified to document the intention to accomplish overprinting.
 - b. Post-slewing must be specified by the NEXT GROUP clause. "NEXT GROUP IS PLUS integer" actually results in integer lines of post-slewing. "NEXT GROUP IS PLUS 0" in connection with "LINE NUMBER IS PLUS 0" on the following line causes overprinting to occur. "NEXT GROUP IS PLUS 0" is the only way to accomplish overprinting.
 - c. When a report group is to consist of multiple lines, each line specification must conform with the rules given in note 4a, and the entries defining all of the lines must be subordinate to the overall report group entry.
5. NEXT GROUP, when specified, refers to spacing conditions following the last line of the report group and the next report group described at a level number equal to or less than the report group level number at which NEXT GROUP is written.
6. In Option 3, level-number, either PICTURE or SIZE and either SOURCE, SUM, or VALUE are required clauses in an elementary entry. The presence of a SOURCE SELECTED clause at a group level indicates a SOURCE is not required at the elementary level. The absence of a COLUMN NUMBER clause in an elementary SOURCE IS entry indicates PICTURE or SIZE is not required in that entry.

CLASS

FUNCTION: To indicate the type of data being described.

[; CLASS IS {
ALPHABETIC
NUMERIC
ALPHANUMERIC
AN }]

Notes:

See the notes under CLASS in the Record Description entries.

COLUMN NUMBER

FUNCTION: To indicate the absolute COLUMN NUMBER on the printed page of the left-most character of the elementary item.

[; COLUMN NUMBER IS integer-1]

Notes:

1. COLUMN NUMBER may be given only at the elementary level within a report group.
2. COLUMN NUMBER indicates that this elementary item is presented in the output report group; if COLUMN NUMBER is not indicated, the elementary item though included in the specifications for the report group for control purposes is suppressed when the report group is produced at object time.
3. Within a report group and a particular LINE NUMBER specification, COLUMN NUMBER entries must be indicated from left to right.

COPY

FUNCTION: To duplicate within this report group a description found elsewhere in the source program or contained in a library.

[data-name-1] COPY data-name-2 [FROM LIBRARY] .

Notes:

See the notes under COPY in the Record Description entries.

DATA-NAME

FUNCTION: To specify a name for the data item being described.

[data-name-1]

Notes for Report Section:

1. If data-name is specified in a level-number entry, the data-name must be the first word following the level-number and must not be qualified or subscripted.
2. Qualification of a data-name can be provided through higher level data-names. Thus, a data-name need not be unique within or between Record Descriptions or Report Group Descriptions provided a higher level name can be used for qualification.
3. Data-name-1 must be given in the following cases:
 - a. When the data-name represents a report group to be referred to by a GENERATE or a USE statement in the Procedure Division.
 - b. When reference is to be made to the SUM counter in the Procedure Division or Report Section.
 - c. When the SELECTED option is included with the SOURCE clause at a higher level to indicate at this lower level the SOURCE data-name(s) which are to be used as elementary items.
4. Since data-name-1 cannot be referred to by the user, except as noted above, and is not meaningful to the Report Writer function in any other case, including data-name-1 with every level number entry is not necessary.

EDITING CLAUSES

FUNCTION: To permit suppression of non-significant zeroes and commas, to permit floating dollar signs or check protection, and to permit the blanking of an item when its value is zero.

$$\left[; \left\{ \begin{array}{l} \underline{\text{ZERO SUPPRESS}} \\ \underline{\text{CHECK PROTECT}} \\ \underline{\text{FLOAT DOLLAR SIGN}} \end{array} \right\} \left[\underline{\text{LEAVING}} \text{ integer PLACES} \right] \left[\underline{\text{BLANK WHEN ZERO}} \right] \right]$$

Notes:

See the notes under Editing Clauses in the Record Description entries.

GROUP INDICATE

FUNCTION: To indicate that this elementary item is to be produced only on the first occurrence of the item after any CONTROL or PAGE break.

[; GROUP INDICATE]

Notes:

1. GROUP INDICATE must only be given at the elementary level within a TYPE DETAIL report group.
2. An elementary item is not only GROUP INDICATED in the first DETAIL report group after a CONTROL break, but is also GROUP INDICATED in the first DETAIL report group of a new PAGE even though a CONTROL break did not occur.

JUSTIFIED

FUNCTION: To specify non-standard positioning of a data item when less than the maximum number of characters may be present.

[; { JUSTIFIED } RIGHT]
 { JUST }

Notes:

See the notes under JUSTIFIED in the Record Description entries.

LEVEL-NUMBER

FUNCTION: To show the hierarchy of items within a report group.

level-number (1 to 49)

Notes for Report Section Only:

1. A level number is required as the first element in each Report Group Description Entry.
2. Level Numbers must start at 01 for report groups, since report groups are the most inclusive groups possible. Less inclusive report groups or elements of report groups are assigned higher (not necessarily successive) level numbers not greater in value than 49.
3. The combination of a level number and a TYPE clause identifies a report group entry; this entry may be one print line or several print lines dependent on format structure within the report group entry.
4. The level number indicates the depth of the particular report group to be GENERATED as output; TYPE indicates the time for generation of this report group.
5. The elementary level number(s) within a report group indicate the elementary item(s) with accompanying PICTURE or SIZE, either SOURCE, SUM or VALUE, and its horizontal position within the report group, if specified, i.e., COLUMN NUMBER.

LINE NUMBER

FUNCTION: To indicate the absolute or relative LINE NUMBER of this entry in reference to the PAGE or the previous entry.

[; LINE NUMBER IS { integer-2
PLUS integer-3
NEXT PAGE }]

Notes:

1. Integer-2 indicates an absolute LINE NUMBER which sets the LINE-COUNTER to this value for printing the item in this entry and following entries within the report group until a different value for the LINE-COUNTER is specified.
2. Integer-3 indicates a relative LINE NUMBER which increments the LINE-COUNTER for printing the item in this entry and following entries within the report group until a different value for the LINE-COUNTER is specified.
3. Integer-2 must be within the range specified by the PAGE LIMITS clause in the Report Name Description Entry.
4. LINE NUMBER must be given for each indicated report group according to the rules given under Report Group Complete Entry Skeleton.
5. If LINE NUMBER is specified for a report group, the entire report group is presented on the specified LINE NUMBER. If LINE NUMBER is specified for a group item, it applies to all subordinate items, and no lower level LINE NUMBER clause is permitted.
6. If NEXT PAGE is specified, this entry will occur on the next page.

NEXT GROUP

FUNCTION: To indicate the spacing to follow the last line of the report group.

[; NEXT GROUP IS { integer-4
PLUS integer-5
NEXT PAGE }]

Notes:

1. Integer-4 cannot exceed the maximum number of lines specified per report PAGE.
2. Integer-4 indicates an absolute LINE NUMBER which sets the LINE-COUNTER to this value after producing the last line of the current report group.
3. Integer-5 indicates a relative LINE NUMBER which increments the LINE-COUNTER by the integer-5 value. Integer-5 represents the number of lines skipped following the last line of the current report group. Further spacing is specified by the LINE NUMBER clause of the next report group produced.
4. NEXT PAGE indicates an automatic skip to the NEXT PAGE following the generation of the last line of the current report group. Appropriate TYPE PAGE/OVERFLOW HEADINGS and TYPE PAGE/OVERFLOW FOOTINGS will be produced as specified when NEXT PAGE is given. The skip to NEXT PAGE after generation of a control footing report group becomes automatic only if the report group is the highest level of control footing to be printed for that control break.
5. The NEXT GROUP clause may appear in conjunction with a TYPE clause.

PICTURE

FUNCTION: To indicate a detailed format of an elementary item, the general characteristics of the item and special editing features for printing.

{ PICTURE } IS (any allowable combination of the characters and
 PIC symbols described in the PICTURE clause under
 Record Description entries)

Notes:

See the notes under PICTURE in the Record Description entries.

RESET

FUNCTION:

The RESET clause indicates the CONTROL data-name that causes the SUM counter in the elementary item entry to be reset to zero on a CONTROL break.

RESET ON { data-name-1 }
FINAL

Notes:

1. Data-name-1 must be one of the data-names described in the CONTROL clause in the Report Description entry. Data-name-1 must be a higher level CONTROL data-name than the CONTROL data-name associated with the CONTROL FOOTING report group in which the SUM and RESET clauses appear.
2. The RESET clause may only be used in conjunction with a SUM clause at the elementary level.
3. After presentation of the TYPE CONTROL FOOTING report group, the counters associated with the report group are reset automatically to zero unless an explicit RESET clause is given specifying reset based on a higher level control than the associated control for the report group.
4. The RESET clause may be used for progressive totaling of data-names where subtotals of data-names may be desired without automatic re-setting upon producing the report group.

SOURCE-SUM-VALUE

FUNCTION: To define the source of this report item.

```
{ ; SOURCE IS [ SELECTED ] data-name-1
; SUM data-name-2 [ UPON data-name-3 ]
; VALUE IS literal-1 }
```

1. Data-name-1, 2, indicate items in either the File, Working-Storage or Constant Section, or is the name of a SUM counter in the Report Section.
2. SOURCE (without SELECTED), SUM and VALUE can only be given at the elementary level. SOURCE IS SELECTED can only be given at a group level.

SOURCE

3. This clause indicates a data-name item which is to be used as the SOURCE for this report item. The item is presented according to the PICTURE or SIZE clause in the associated elementary report group entry.
4. When the SELECTED option is given, data-name-1 represents a group item. Data-name-1 may be qualified but not subscripted. No subordinate groups may have a SELECTED clause. The data-name(s) described at the elementary level in the source group then become SOURCE data-name entries in the associated report group. The SELECTED elementary level data-name(s) must be unique data-name(s). They may be subscripted.
5. The elementary level items within data-name-1 are matched against the data-name(s) specified at the elementary level within the report group. Matching data-name(s) are SELECTED as SOURCE item entries to be included and presented within the report group, according to the PICTURE or CLASS and SIZE specifications given with the data-name in the report group entry.

SUM (See the notes under the ADD verb.)

6. A SUM clause may only appear in a TYPE CONTROL FOOTING report group.

7. If a SUM counter needs to be referred to by a Procedure Division statement or Report Section entry, a data-name must be specified with the SUM clause entry. The data-name then represents a summation counter automatically generated by the Report Writer to total the operand specified immediately following SUM. If a summation counter is never referred to, the counter need not be named explicitly by a data-name entry.
8. Whether the SUM clause names the summation counter or not, the Report Writer generates an appropriate NUMERIC storage item based on the numeric characters to be presented within the associated PICTURE or SIZE clause.
9. Each data-name-2 item being summed, must appear as a SOURCE item in a TYPE DETAIL report group in the current Report Description or must be names of SUM counters in a TYPE CONTROL report group at an equal or lower position in the control hierarchy. Although the item(s) must be explicitly written in a TYPE DETAIL report group, they may actually be suppressed at presentation time. In this manner, direct association without ambiguity can be made from the current data available by a GENERATE statement to the data items to be presented within the REPORT Section.
10. Data-name-2 may be subscripted for selective summation and must be qualified to make the named item unique.
11. If higher level report groups are indicated in the CONTROL hierarchy, counter updating procedures commonly called "rolling counters forward," take place prior to the reset operation.
12. Updating is automatic between adjacent level TYPE CONTROL FOOTING report groups. The summation of data items is accomplished explicitly or implicitly at every TYPE CONTROL FOOTING level in order for this automatic procedure to function; e.g., if a minor SUM CONTROL of a data item is not desired for presentation but the intermediate and major SUM CONTROLS are, the minor SUM specification may be omitted with the Report Writer automatically generating an appropriate summation counter for the updating function.

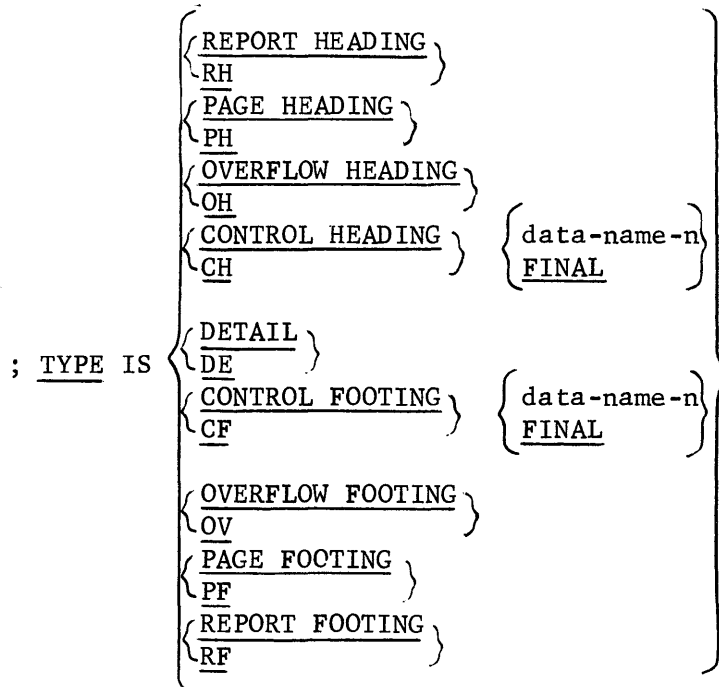
13. The UPON data-name-3 option is required to obtain selective summation for a particular data item which is named as a SOURCE item in two or more TYPE DETAIL report groups. Data-name-3 must be the name of a TYPE DETAIL report group. Data-name-2 must be a SOURCE data-item in data-name-3. If the UPON data-name-3 option is not used, data-name-2 is added to the SUM counter at each execution of a GENERATE statement. This statement generates a TYPE DETAIL report group that contains the SUM operand at the elementary level.

VALUE

14. Literal-1 may be NUMERIC, ALPHANUMERIC, ALPHABETIC or a figurative constant.

TYPE

FUNCTION: To specify the circumstances under which this report group is to be presented.



Notes:

1. REPORT HEADING or RH indicates a report group that is produced only once at the initiation of a report. There may be only one 01 level report group of this TYPE in a report.
2. PAGE HEADING or PH indicates a report group that is produced at the beginning of each page according to PAGE and OVERFLOW condition rules as specified in Note 18. There may be only one 01 level report group of this TYPE in a report.
3. OVERFLOW HEADING or OH indicates a report group that is produced at the beginning of a page following an OVERFLOW condition according to PAGE and OVERFLOW rules as specified in Note 18. There may be only one 01 level report group of this TYPE in a report.
4. CONTROL HEADING or CH indicates a report group that is produced at the beginning of a control group for a designated data-name or in the case of FINAL is produced once at the initiation of a report before the first control group. There may be only one 01 level report group of this TYPE for each CONTROL data-name and for FINAL specified in a report.

5. **DETAIL** or **DE** indicates a report group that may be referenced by **GENERATE** statements in the Procedure Division. Each **DETAIL** report group must have a data-name. If the report group **TYPE** is other than **DETAIL**, the Procedure Division statement, **GENERATE** data-name, directs the report writer to produce the named report group.
6. **CONTROL FOOTING** or **CF** indicates a report group that is produced at the end of a control group for a designated data-name or is produced once at the termination of a report ending a **FINAL** control group. There may be only one 01 level report group of this **TYPE** for each **CONTROL** data-name and for **FINAL** specified in a report.
7. **OVERFLOW FOOTING** or **OV** indicates a report group that is produced at the bottom of a page following an **OVERFLOW** condition according to **PAGE** and **OVERFLOW** rules as specified in Note 18. There may be only one 01 level report group of this **TYPE** in a report.
8. **PAGE FOOTING** or **PF** indicates a report group that is produced at the bottom of each page according to **PAGE** and **OVERFLOW** condition rules as specified in Note 18. There may be only one 01 level report group of this **TYPE** in a report.
9. **REPORT FOOTING** or **RF** indicates a report group that is produced only once at the termination of a report. There may be only one 01 level report group of this **TYPE** in a report.
10. Data-name-n, as well as **FINAL**, must be one of the data-name(s) described in the **CONTROL(S)** clause in the Report Name Description Entry.
11. A **FINAL TYPE CONTROL** break may be designated only once for **CONTROL HEADING** and/or **CONTROL FOOTING** within a particular report.
12. Nothing will precede a **REPORT HEADING** and nothing will follow a **REPORT FOOTING** within a report.
13. The **HEADING** and/or **FOOTING** report groups occur in the following Report Writer sequence, if all exist for a given report:

```
REPORT HEADING (one occurrence only first page)
PAGE HEADING or OVERFLOW HEADING
:
CONTROL HEADING
DETAILS
CONTROL FOOTING
:
PAGE FOOTING or OVERFLOW FOOTING
REPORT FOOTING (one occurrence only last page)
```

14. CONTROL HEADING report groups are presented in the following hierarchical arrangement:

```

FINAL CONTROL HEADING
MAJOR CONTROL HEADING
      :
MINOR CONTROL HEADING

```

CONTROL FOOTING report groups are presented in the following hierarchical arrangement:

```

MINOR CONTROL FOOTING
      :
MAJOR CONTROL FOOTING
FINAL CONTROL FOOTING

```

15. CONTROL HEADING report groups appear with the current values of any indicated SOURCE data items before the DETAIL report groups of the CONTROL group are produced. CONTROL FOOTING report groups appear with the previous values of any indicated CONTROL data-name items just after the DETAIL report groups of that CONTROL group have been produced. These report groups appear whenever a CONTROL break is noted. LINE NUMBER determines the absolute or relative position of the CONTROL report groups exclusive of the other HEADING and FOOTING report groups.
16. The concept of the OVERFLOW condition in a Report Writer is based on the logical definition of a page format relative to the presentation of a complete control group. For purposes of the OVERFLOW condition, a complete control group depends on the change of a data item value within a designated order of specific data items. If the change is a MINOR control group break, the complete control group includes the HEADING, DETAIL and FOOTING report groups associated with the MINOR CONTROL specification. If the change is a MAJOR control group break, the complete control groups includes the HEADING, DETAIL and FOOTING report groups associated with the MINOR, INTERMEDIATE, and MAJOR CONTROL specifications, etc. Thus, during process time, if a page format does not allow a complete control group to be presented within the definition of the page, an OVERFLOW condition is said to exist from the last DETAIL report group printed in the control group on one page to the first report group printed in the control group on the next page. Between the "points" of "from and to" described above, OVERFLOW FOOTING and OVERFLOW HEADING report groups may be produced, if specified. If a complete control group, as described above, and none of the next control group can be presented within the definition of the page, a PAGE condition is said to exist from the last DETAIL report group and therefore, PAGE FOOTING and PAGE HEADING report groups are produced, if specified.

17. PAGE HEADING and OVERFLOW HEADING, and PAGE FOOTING and OVERFLOW FOOTING, if specified in a report, are mutually exclusive for any one page. The absence of a TYPE OVERFLOW HEADING indicates that TYPE PAGE HEADING report group(s), if specified, are produced at the beginning of each page regardless of the condition that prompted the new page. Likewise, the absence of a TYPE OVERFLOW FOOTING indicates that TYPE PAGE FOOTING report group(s), if specified, are produced at the bottom of each page regardless of the condition that ended the current page.

18. In order to recognize the OVERFLOW condition within the Report Writer the PAGE LIMITS clause must be given including the LAST DETAIL option. If both TYPE PAGE HEADING and OVERFLOW HEADING and/or TYPE PAGE FOOTING and OVERFLOW FOOTING report group(s) are specified in the same report and if the LINE-COUNTER will exceed the LAST DETAIL limit for generation of the current report group, the following rules apply:
 - a. Without the PAGE LIMITS FOOTING option, if the current DETAIL report group is not the first DETAIL report group of a new CONTROL group, an OVERFLOW condition exists from this position on the page to the position on the next page where the FIRST DETAIL report group can be presented. If the current DETAIL report group is the first DETAIL report group of a new CONTROL group, a PAGE condition exists.

 - b. With the PAGE LIMITS FOOTING option, if the current report group is a TYPE DETAIL report group, an OVERFLOW condition exists as stated in a. above. If the current report group is a CONTROL FOOTING report group, the test is made to determine if the LINE-COUNTER will exceed the FOOTING limit for generation of the complete CONTROL FOOTING report group. If all the report group(s) associated with this CONTROL break can be produced within the limit specified, a PAGE condition exists following the CONTROL FOOTING report group. If all the report group(s) associated with this CONTROL break cannot be produced within the limit specified, an OVERFLOW condition exists which means the TYPE CONTROL FOOTING report group(s) are produced on the following page.

 - c. Without the PAGE LIMITS LAST DETAIL option, an OVERFLOW condition cannot exist within the Report Writer. Therefore, with or without the PAGE LIMITS FOOTING option, TYPE PAGE FOOTINGS, as differentiated from TYPE OVERFLOW FOOTINGS described in Note 18, are the only report groups that are produced, if specified, after TYPE DETAIL and TYPE CONTROL report groups on a page. The absence of both LAST DETAIL and FOOTING options invalidates the presence of either OVERFLOW or PAGE FOOTING report groups.

- d. The careful programmer by setting proper PAGE LIMITS can effectively control the format within a page using all the TYPE report groups available. Omitting the FOOTING option indicates the Report Writer always functions as described in a. above. However, for the programmer who desires to produce all control footing report group(s) on one page if a CONTROL break does occur, the Report Writer is able to function as described in b. above.
 - e. The rules stated in Note 17 above apply regardless of the conditions that may be recognized by the Report Writer as described in Note 18.
19. The TYPE clause must occur at the 01 level.

USAGE

FUNCTION: To specify the dominant use of a data item.

[; USAGE IS { DISPLAY
DISPLAY-1 }]

Notes:

See the notes under USAGE in the Record Description Entries. The only allowable USAGES for report items are DISPLAY and DISPLAY-1.

SUMMARYFILE SECTION

The FILE SECTION contains a section header, File Description entries and Record Description entries for label records. Some of the information about the file may be in the COBOL library and therefore may not appear explicitly in the FILE Section. The order of information is as follows:

```

FILE SECTION.
FD file-name...
01 label-name...
      :
      :
01 record-name...
      :
      :
FD file-name...
      :
      :
SD file-name...

```

Specifications and Handling of Labels

The COBOL System provides for the automatic handling of four types of labels - beginning and ending, file and tape. The notes for the LABEL RECORDS clause in the File Description entry contain an indication of the relative position of these labels on the tape(s) associated with a file.

A label record is a logical record containing the labeling information about a tape or file. There are many different types of label records, but some of these are fairly standard. In order to have common recognition of these records, fixed names have been assigned. The label records with fixed names are:

```

BEGINNING-TAPE-LABEL
BEGINNING-FILE-LABEL
ENDING-FILE-LABEL
ENDING-TAPE-LABEL

```

A Record Description must be available for each label record employed. Record Descriptions for label records are prepared in the same manner as those prepared for data records; however, since the input/output system must perform special operations on certain items within the label record, fixed names have been assigned to those label items which have particular significance.

For purposes of discussion, label items are classified as follows:

1. Those which must contain unique values depending on the particular file involved, such as the name and number of the file.

2. Those which have a general meaning in the processing of files, such as record count, tape number, etc.
3. Those having special functions which are not handled automatically.

An item which must have a unique value depending on the particular file is handled in the following manner:

1. In preparing the Record Description entry, any name may be assigned to the item. Since the value of the item is a variable, it is not shown.
2. In preparing the File Description entry, the name of the item and the value which it must contain is listed in the VALUE clause.
3. In processing an input file, an equality test is made between the contents of the label item, and the corresponding value specified in the file description.
4. In processing an output file, the value specified in the file description is entered in the label item of the beginning file.

The standard GE-600 Series BEGINNING-TAPE-LABEL and BEGINNING-FILE-LABEL have identical formats:

```

01 BEGINNING-TAPE-LABEL; SIZE 84 DISPLAY CHARACTERS.
02 LABEL-IDENTIFIER; PICTURE X(12) VALUE
   IS "GE 600 BTL ".
02 INSTALLATION; PICTURE X(6).
02 REEL-SERIAL-NUMBER; PICTURE B9(5).
02 FILE-SERIAL-NUMBER; PICTURE B9(5).
02 REEL-NUMBER; PICTURE BB9999; RANGE
   IS " 0001" THRU " 9999".
02 DATE-WRITTEN; SIZE 6.
03 LABEL-YEAR: PICTURE B99.
03 LABEL-DAY: PICTURE 999; RANGE IS 001
   THRU 365.
02 FILLER; PICTURE XXX; VALUE SPACES.
02 RETENTION-PERIOD; PICTURE 999;
   RANGE IS 001 THRU 999.
02 IDENTIFICATION; PICTURE X(12).
02 FILLER; SIZE 24 AN CHARACTERS.
66 ID RENAMES IDENTIFICATION.
  
```

The standard GE-600 Series ENDING-TAPE-LABEL and ENDING-FILE-LABEL also have identical formats:

```

01 ENDING-TAPE-LABEL; SIZE 84.
02 SENTINEL; PICTURE X(6).
   88 END-OF-TAPE; VALUE IS " EOR ".
   88 END-OF-FILE; VALUE IS " EOF ".
02 BLOCK-COUNT; PICTURE 9(6).
02 FILLER; SIZE IS 72 AN CHARACTERS.
  
```

Except when RECORDING MODE IS BCD, BLOCK-COUNT has USAGE COMPUTATIONAL-1. For BCD files, it is DISPLAY, as is implicitly shown above.

The FILLER items at the end of each of the above formats may be replaced by descriptions of additional data items to be included in the label records. Such additional items, if present, must be processed in USE procedures. Unless such items are included, the STANDARD label descriptions are implicitly described by the LABEL RECORDS ARE STANDARD clause, and consequently STANDARD labels need not be described, even if their contents are referenced in USE procedures.

Label record contents may be accessed only by USE procedures, and only one label record is available when USE procedures are executed. Standard label item data-names need never be qualified in procedural references.

If explicit label record descriptions are specified, the standard contents must be described exactly as shown above, with respect to data-names, PICTURES, and position. The overall size of each label record must be 84 characters. Departures from these rules can lead to unpredictable results. VALUE clauses do not result in automatic MOVES of the specified literals to output labels. All standard label items are automatically handled by the input/output routines; an option is provided for specifying the literals to be used with IDENTIFICATION and RETENTION-PERIOD via the FD VALUE clause.

Because label record formats are generally not the same for different computer lines, a program with explicit label record descriptions must usually be modified if it is to be compiled on any computer line other than that for which it was originally programmed. Not all fixed label item data-names can be used with all compilers, but the fixed label record names themselves are usually available, in addition to the following fixed data-names:

IDENTIFICATION
REEL-NUMBER
DATE-WRITTEN
BLOCK-COUNT
SENTINEL
END-OF-TAPE
END-OF-FILE

If label record descriptions are explicitly given for a file, they must precede the descriptions of logical data records. Standard label record names must never be mentioned in a DATA RECORD[S] clause.

WORKING-STORAGE SECTION

Working Storage is that part of computer memory set aside for intermediate processing of data. The difference between WORKING-STORAGE and FILE storage is that the former deals with computer memory requirements for the storage of intermediate data results whereas the latter deals with the computer memory requirements for the storage of each record of the file.

Organization

Whereas the FILE SECTION is composed of File Description entries and Record Description entries, the WORKING-STORAGE SECTION is composed only of Record Description entries. The WORKING-STORAGE SECTION begins with a section-header and a period, followed by Record Description entries for non-contiguous working storage items, and then by Record Description entries for working storage records, in that order. The skeletal format for WORKING-STORAGE SECTION is as follows:

```

WORKING STORAGE SECTION.
  77 data-name-1
    88 condition-name-1
      :
      :
  77 data-name-n
  01 data-name-2
    02 data-name-3
      :
      :
  66 data-name-m RENAMES data-name-3
  01 data-name-4
    02 data-name-5
      03 data-name-n
        88 condition-name-2

```

Non-Contiguous Working-Storage

Items in working storage which bear no relationship to one another need not be grouped into records, provided they do not need to be further subdivided. Such items are called non-contiguous items. Each of these items is defined in a separate Record Description entry which begins with the special level number 77.

The following Record Description clauses are required in each level 77 entry:

1. level-number
2. data-name
3. SIZE (when PICTURE is not specified).

The OCCURS and REDEFINES clauses are not meaningful in level 77 entries and will cause an error at compilation time if used. Other Record Description clauses are optional and can be used to complete the description of the item, if necessary.

Working-Storage Records

Data elements in working storage which bear a definite relationship to one another must be grouped into records according to the rules for formation of Record Descriptions. All clauses which are used in normal input or output Record Descriptions can be used in a WORKING-STORAGE Record Description, including REDEFINES, OCCURS, and COPY. Each working storage record-name (01 level) must be unique since it cannot be qualified by a file-name or section-name. Subordinate data-names need not be unique if they can be made unique by qualification.

Initial Values

The initial value of any item in the WORKING-STORAGE SECTION may be specified by using the VALUE clause of the Record Description. If VALUE is not specified the initial values may be unpredictable. VALUE can be specified only in terms of homogeneous characters (i.e., characters having the same USAGE). VALUE, therefore, cannot be specified in a group item which contains elementary items having different USAGES. All the rules for the expression of literals and figurative constants apply. The size of a literal used to specify an initial value can be equal to or less than the size specified in the SIZE clause (or PICTURE) of the associated data entry, but it cannot be greater. When the size is less, normal rules for a MOVE of a literal apply.

For example:

1. 77 PAGE-1; SIZE IS 7 NUMERIC COMPUTATIONAL CHARACTERS
VALUE IS 0000342. (Legal)
2. 04 PAGE-NO; SIZE 4; CLASS NUMERIC; VALUE IS 5. (Legal)
3. 02 ADDRESS-1; SIZE 5; CLASS AN VALUE IS "XY1245Z". (Illegal)
4. 03 GROUPING; SIZE 9; CLASS NUMERIC; VALUE IS ZEROS. (Legal)

In example 2, the CLASS is NUMERIC. Therefore, the value of the item will be right justified with zero fill and will be stored as 0005.

Condition-Names

Any working storage item may constitute a conditional variable with which one or more condition-names may be associated. Entries defining condition-names must immediately follow the item to which they relate. Both the conditional variable entry and the associated condition-name entries may contain VALUE clauses. (See VALUE clause for examples.)

CONSTANT SECTION

The concept of literals and figuratives enables the user to specify the value of a constant by writing its actual value (or a figurative representation of that value). It is often desirable to name this value and then refer to it by its name. For example:

6%(.06) may be named as INTEREST-RATE and then referred to by its name (INTEREST-RATE) instead of its value (.06).

Constant Storage is computer memory area which is set aside to save named constants for use in a given program.

Organization

The CONSTANT SECTION is organized in exactly the same way as the WORKING-STORAGE SECTION, beginning with a section header, followed by Record Description entries for non-contiguous constants, and then by Record Description entries, in that order. The skeletal format for the CONSTANT SECTION is as follows:

```

CONSTANT SECTION.
  77 data-name-1
      :
  77 data-name-n
  01 data-name-2
    02 data-name-3
      :
  01 data-name-4
    02 data-name-5
      03 data-name-6

```

Non-Contiguous Constant Storage

Constants which bear no relationship to one another need not be grouped into records provided they do not need to be further subdivided. Such items are called non-contiguous constants. Each of these constants is defined in a separate Record Description entry which begins with the special level number 77.

The following Record Description clauses are required in each level 77 entry:

1. level number
2. data-name
3. SIZE (when PICTURE is not given)
4. VALUE

The OCCURS and REDEFINES clauses are not meaningful for level 77 entries and will cause an error at compilation time if used. Other Record Description clauses are optional and can be used to complete the description of the constant when necessary.

Constant Records

Named constants in CONSTANT SECTION which bear a definite relationship to one another must be grouped into records according to the rules for formation of Record Descriptions. All Record Description clauses can be used in a Constant Record Description, including REDEFINES, OCCURS, and COPY. Each CONSTANT SECTION record-name (01 level) must be unique since it cannot be qualified by a file-name or section-name. Subordinate data-names need not be unique if they can be made unique by qualification.

VALUE of Constants

In the definition of constants, the VALUE clause is required. VALUE cannot be specified in a group entry which contains items having different USAGES. All the rules for the expression of literals and figurative constants apply. The size of a literal used to specify the value of a constant can be equal to or less than the specified size of the item, but it cannot be greater.

Condition-Names

Since a constant can have only one value, there can be no associated condition-names. The use of a condition-name entry (level 88) in the CONSTANT SECTION is, therefore, illegal and will constitute an error in the source program.

Table of Constants

Tables of constants to be referenced by means of subscripting are defined in one of the following ways:

1. The table may be described as a record by a set of contiguous Record Description entries each of which specifies the VALUE of an element, or part of an element, of the table. In defining the record and its elements, any Record Description clause (i.e., SIZE, USAGE, PICTURE, editing information, etc.) may be used to complete the definition where required. This form is required when the elements of the table require separate handling due to SYNCHRONIZATION, USAGE, etc. The structure of the table is then shown by use of the REDEFINES entry and its associated subordinate entries.
2. When the elements of the table do not require separate handling, the VALUE of the entire table may be given in the entry defining the entire table. The lower level entries will show the hierarchical structure of the table.

VII. PROCEDURE DIVISION

GENERAL DESCRIPTION

The PROCEDURE DIVISION contains the procedures needed to solve a given problem. These procedures are written as sentences, combined to form paragraphs, which in turn may be combined to form sections.

RULES OF PROCEDURE FORMATION

COBOL procedures are expressed in a manner similar (but not identical) to normal English prose. The basic unit of procedure formation is a sentence or a group of successive sentences. A procedure is a paragraph, or a group of successive paragraphs, or a section, or a group of successive sections within the PROCEDURE DIVISION.

STATEMENTS

There are three types of statements: imperative statements, conditional statements, and compiler directing statements.

Imperative Statements

An imperative statement consists of either a verb (excluding compiler directing verbs) and its operands, or a sequence of imperative statements.

Conditional Statements

A conditional statement is defined to be one of the four following forms:

1. IF condition $\left\{ \begin{array}{l} \text{Statement-1} \\ \underline{\text{NEXT SENTENCE}} \end{array} \right\} \left\{ \begin{array}{l} \underline{\text{OTHERWISE}} \\ \underline{\text{ELSE}} \end{array} \right\} \left\{ \begin{array}{l} \text{Statement-2} \\ \underline{\text{NEXT SENTENCE}} \end{array} \right\}$
2. Any arithmetic verb; ON SIZE ERROR any imperative statement.
3. $\left\{ \begin{array}{l} \underline{\text{READ}} \\ \underline{\text{RETURN}} \end{array} \right\}$ file-name RECORD $\left[\underline{\text{INTO}} \text{ data-name} \right] \left[; \text{ AT } \underline{\text{END}} \text{ any imperative statement.} \right]$
4. Any imperative statement not containing a GO or STOP RUN statement followed by a conditional statement of the form 1, 2, or 3.

The "any arithmetic verb" in form 2 is any of the verbs ADD, SUBTRACT, MULTIPLY, DIVIDE or COMPUTE. Form 3 is considered conditional whether or not the AT END clause is explicitly written, since it must be applied to the READ statement either implicitly or explicitly.

Statement-1 or statement-2 can be either imperative or conditional; and if conditional can, in turn, contain conditional statements in arbitrary depth. If statement-1 or statement-2 is conditional, then the conditions within the conditional statement are considered to be "nested."

Compiler Directing Statements

A compiler directing statement consists of a compiler directing verb and its operands.

SENTENCES

A sentence consists of a sequence of one or more statements, the last of which is terminated by a period. The statements comprising the sentence must be either (a) one Compiler Directing Statement, or (b) an Imperative or Conditional Statement, syntactically correct according to the above rules.

A sentence which is composed of a Compiler Directing Statement is called a "Compiler Directing Sentence." A sentence which is composed of an Imperative or a Conditional Statement is called a "Procedural Sentence."

Imperative Sentences

An imperative statement terminated by a period is an imperative sentence.

EXAMPLE: MOVE A TO B.
MOVE A TO B; ADD C TO D.

An imperative sentence can contain either a GO statement or a STOP RUN statement, which (if present) must be the last statement in the sentence.

EXAMPLE: MOVE A TO B; ADD C TO D THEN GO TO START.

Conditional Sentences

A conditional statement terminated by a period is a conditional sentence.

EXAMPLE:

IF X EQUALS Y THEN MOVE A TO B; OTHERWISE MOVE C TO D.

IF X EQUALS Y MOVE A TO B; IF W EQUALS T ADD A TO B;
OTHERWISE NEXT SENTENCE; OTHERWISE MOVE C TO D.

MOVE Y TO A; OPEN OUTPUT ERROR-FILE; ADD X TO Y;
ON SIZE ERROR ADD Y TO Z; DISPLAY "OVERFLOW ON Y".

READ ERROR-RECORD; AT END GO TO CLOSING-ROUTINE.

If the phrase "OTHERWISE NEXT SENTENCE" immediately precedes the period, then the phrase "OTHERWISE NEXT SENTENCE" may be eliminated. This rule may then be applied again to the resulting sentence.

Compiler Directing Sentences

A compiler directing statement terminated with a period is a compiler directing sentence.

EXAMPLE: USE AFTER ERROR PROCEDURE ON MASTER-FILE.

SENTENCE PUNCTUATION

Verb Formats

Punctuation rules for individual verbs are as shown in verb formats and in Chapters IV and V.

Sentence Formats

The following rules apply to the punctuation of sentences:

1. A sentence is terminated by a period.
2. A separator is a word or character used for the purpose of enhancing readability. Use of a separator is optional.
3. The allowable separators are:

;
THEN

4. These separators must not be followed immediately by another such separator.
5. Separators may be used in the following places:
 - a. Between statements.
 - b. In a conditional statement:

Between the condition and statement-1

Between statement-1 and OTHERWISE.

SENTENCE EXECUTION

For the following discussion, by "execution of a sentence or a statement within a sentence" is meant "execution of an object program compiled from a sentence, or from a statement within a sentence which has been written in COBOL." By "transfer of control" is meant "transfer of control in the object program by transferring (GOing) from one place (control point) to another place (control point) out of the written sequence." By "passing of control" is meant "passing of control in the object program by passing from one place (control point) to the next place (control point) in the written sequence."

Whenever a GO statement is encountered during the execution of a sentence or statement, there will be an unconditional transfer of control to the first procedural sentence of the paragraph or section referenced by the GO statement.

Imperative Sentences

An imperative sentence is executed in its entirety and control is passed to the next procedural sentence.

Conditional Sentences

IF condition { statement-1
NEXT SENTENCE } { OTHERWISE
ELSE } { statement-2
NEXT SENTENCE }

In the conditional sentence above, the condition is an expression which is true or false. If the condition is true, then statement-1 is executed and control is transferred to the next sentence. If the condition is false, statement-2 is executed and then control is passed to the next sentence.

If statement-1 is conditional, then the conditional statement should be the last (or only) statement comprising statement-1. For example, the conditional sentence would then have the form:

IF condition-1 imperative-statement-1 IF condition-2 statement-3
OTHERWISE statement-4 OTHERWISE statement-2

If condition-1 is true imperative-statement-1 is executed, then if condition-2 is true statement-3 is executed and control is transferred to the next sentence. If condition-2 is false then statement-4 is executed and control is transferred to the next sentence. If condition-1 is false statement-2 is executed and control is passed to the next sentence.

Statement-3 can in turn be either imperative or conditional, and if conditional can in turn contain conditional statements in arbitrary depth. In an identical manner statement-4 can be either imperative or conditional, as can statement-2.

The execution of the phrase "OTHERWISE NEXT SENTENCE" causes a transfer of control to the next sentence as written, except when it appears in the last sentence of a procedure being PERFORMed, in which case control is passed to the return mechanism.

Compiler Directing Sentences

Compiler Directing Sentences direct a COBOL processor to take action at compilation time, rather than specifying action to be taken by the object program.

CONTROL RELATIONSHIP BETWEEN PROCEDURES

In COBOL, Imperative and Conditional sentences describe the procedure that is to be accomplished. The sentences are written successively, according to the rules of the Reference Format (Chapter III) to establish the sequence in which the object-program is to execute the procedure.

In executing procedures, control is transferred only to the beginning of a paragraph. Control is passed to a sentence within a paragraph only from the sentence written immediately preceding it. If a procedure is named, control can be passed to it from the sentence immediately preceding it, or can be transferred to it from any sentence which contains a GO TO or PERFORM followed by the name of the procedure to which control is to be transferred.

CONDITIONALS

GENERAL DESCRIPTION

Conditional procedures are one of the keystones in describing data processing problems. COBOL makes available to the programmer several means of expressing conditional situations.

COBOL conditionals generally involve the key word IF followed by the conditions to be examined followed by the operations to be performed. Depending upon the truth or falsity of the conditions different sets of operations are to be performed.

CONDITIONS

Simple Conditions

A simple condition is one of three types of tests. These tests and the acceptable formats for stating them are described below. (The word IF is not part of the conditional, but is shown in the formats of this section to improve readability.)

1. Relation Tests

A relation test involves a comparison of two operands; either of these two operands can be a data-name, a literal, or a formula. The comparison of two literals is not permitted. Throughout the remainder of the discussion the word "item" means either a data item or a literal. Comparison of NUMERIC items is permitted regardless of their individual USAGES. All other comparisons require that the USAGE of the items being compared be the same.

a. Comparison of NUMERIC Items

For NUMERIC items, a comparison results in the determination that the value of one of the items is LESS THAN, EQUAL TO, or GREATER THAN the other.

The comparison of NUMERIC items is based on the respective values of the items considered purely as algebraic values. The item length, in terms of the number of digits, is not itself significant. Zero is considered to represent a unique value regardless of the length, sign or implied decimal point location of an item.

b. Comparison of non-NUMERIC Items

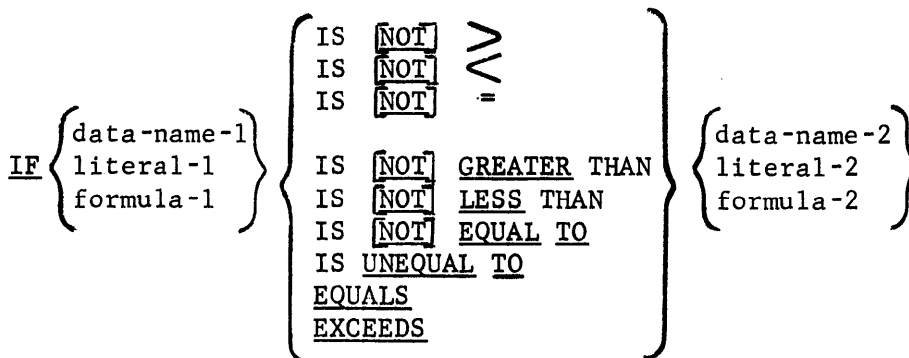
For two non-NUMERIC items, or one NUMERIC (excluding the operational sign) and one non-NUMERIC item, a comparison results in the determination that one of the items is LESS THAN, EQUAL TO, or GREATER THAN the other with respect to an ordered character set. Except when USAGE of the items is DISPLAY-2, the character set order is determined by the Standard Collating Sequence. The order for DISPLAY-2 items is determined by the Commercial Collating Sequence. Literals cannot be compared to DISPLAY-2 items.

If the items are of equal SIZE, comparison proceeds by comparing characters in corresponding character positions starting from the high order end and continuing until either a pair of unequal characters is encountered or the low order end of the item is reached, whichever comes first. The items are determined to be EQUAL when the low order end is reached.

The first encountered pair of unequal characters is compared for relative location in the ordered character set. The item which contains that character which is positioned higher in the ordered sequence is determined to be the GREATER item.

If the items are of unequal SIZE, comparison proceeds as described above. If this process exhausts the characters of the item of lesser SIZE, then the item of lesser SIZE is LESS THAN the item of larger SIZE unless the remainder of the item of larger SIZE consists solely of spaces, in which case the two items are EQUAL.

The format for full relation tests is:



The word EXCEEDS is equivalent to IS GREATER THAN. The phrase IS UNEQUAL TO is equivalent to IS NOT EQUAL TO.

In the preceding format, the actual choice from data-name-1, literal-1, and formula-1 is called the subject. The choice from data-name-2, literal-2, and formula-2 is called the object. The subject and the object cannot both be literals.

STANDARD COLLATING SEQUENCE

Standard Character*	Standard Octal Code	Commercial Octal Code	Punched Card	Standard Character*	Standard Octal Code	Commercial Octal Code	Punched Card
0	00	60	0	↑	40	35	11-0
1	01	61	1	J	41	36	11-1
2	02	62	2	K	42	37	11-2
3	03	63	3	L	43	40	11-3
4	04	64	4	M	44	41	11-4
5	05	65	5	N	45	42	11-5
6	06	66	6	O	46	43	11-6
7	07	67	7	P	47	44	11-7
8	10	70	8	Q	50	45	11-8
9	11	71	9	R	51	46	11-9
	12	16	2-8	-	52	10	11
#	13	17	3-8	\$	53	06	11-3-8
@	14	22	4-8	*	54	07	11-4-8
:	15	72	5-8)	55	03	11-5-8
∨	16	20	6-8	;	56	76	11-6-8
Ignore or? space	17	73	7-8	'	57	21	11-7-8
A	21	00	blank	+	60	23	12-0
B	22	24	12-1	/	61	11	0-1
C	23	25	12-2	S	62	50	0-2
D	24	26	12-3	T	63	51	0-3
E	25	27	12-4	U	64	52	0-4
F	26	30	12-5	V	65	53	0-5
G	27	31	12-6	W	66	54	0-6
H	30	32	12-7	X	67	55	0-7
I	31	33	12-8	Y	70	56	0-8
&	32	34	12-9	Z	71	57	0-9
.	33	05	12	←	72	47	0-2-8
j	34	01	12-3-8	,	73	12	0-3-8
(35	02	12-4-8	%	74	13	0-4-8
^	36	15	12-5-8	=	75	14	0-5-8
/	37	74	12-6-8	"	76	04	0-6-8
		75	12-7-8	!	77	77	0-7-8

*Except for the control characters with standard octal codes 17 and 77, the GE 600-Line printer characters are the same as the graphic symbols in this table.

COMMERCIAL COLLATING SEQUENCE

Commercial Character	Commercial Octal Code	Standard Octal Code	Punched Card	Printer
Space	00	20	blank	Space
) or H	01	33	12-3-8	.
	02	34	12-4-8]
	03	55	11-5-8)
	04	76	0-6-8	"
	05	32	12	&
\$	06	53	11-3-8	\$
*	07	54	11-4-8	*
-	10	52	11	-
/	11	61	0-1	/
, or %	12	73	0-3-8	,
	13	74	0-4-8	%
	14	75	0-5-8	=
# or =	15	35	12-5-8	(
	16	12	2-8	[
' or @	17	13	3-8	#
	20	16	6-8	>
ø	21	57	11-7-8	'
	22	14	4-8	@
	23	60	12-0	+
A	24	21	12-1	A
B	25	22	12-2	B
C	26	23	12-3	C
D	27	24	12-4	D
E	30	25	12-5	E
F	31	26	12-6	F
G	32	27	12-7	G
H	33	30	12-8	H
I	34	31	12-9	I
Ö	35	40	11-0	
J	36	41	11-1	J
K	37	42	11-2	K

Commercial Character	Commercial Octal Code	Standard Octal Code	Punched Card	Printer
L	40	43	11- 3	L
M	41	44	11- 4	M
N	42	45	11- 5	N
O	43	46	11- 6	O
P	44	47	11- 7	P
Q	45	50	11- 8	Q
R	46	51	11- 9	R
≠	47	72	0- 2-8	←
S	50	62	0- 2	S
T	51	63	0- 3	T
U	52	64	0- 4	U
V	53	65	0- 5	V
W	54	66	0- 6	W
X	55	67	0- 7	X
Y	56	70	0- 8	Y
Z	57	71	0- 9	Z
0	60	00	0	0
1	61	01	1	1
2	62	02	2	2
3	63	03	3	3
4	64	04	4	4
5	65	05	5	5
6	66	06	6	6
7	67	07	7	7
8	70	10	8	8
9	71	11	9	9
	72	15	5-8	:
	73	17	7-8	?
	74	36	12-6-8	<
	75	37	12-7-8	/
	76	56	11-6-8	:
	77	77	0-7-8	!

An alternative way of stating a comparison of the value zero with a formula, or with an item whose CLASS is determined to be NUMERIC by a Record Description entry, or NUMERIC by a class test at object time, is provided by the following form:

$$\text{IF } \left\{ \begin{array}{l} \text{data-name} \\ \text{formula} \end{array} \right\} \text{ IS } [\text{NOT}] \left\{ \begin{array}{l} \text{POSITIVE} \\ \text{NEGATIVE} \\ \text{ZERO} \end{array} \right\}$$

An item or formula is POSITIVE only if its value is greater than zero. An item or formula whose value is zero is NOT POSITIVE. An item or formula is NEGATIVE only if its value is less than zero. An item or formula whose value is zero is NOT NEGATIVE. In COBOL, the value zero is considered neither positive nor negative.

2. CLASS Test

The CLASS of any item can be tested as follows:

$$\text{IF data-name IS } [\text{NOT}] \left\{ \begin{array}{l} \text{NUMERIC} \\ \text{ALPHABETIC} \end{array} \right\}$$

The NUMERIC test may be used to check the validity of the contents of a NUMERIC DISPLAY item (SIGNED or not). In such a test, the contents of the item will be tested for conformity with the definition of the CLASS.

3. Conditional Variable Test

A conditional variable test is one in which an item is tested to determine whether or not one of the values associated with a condition-name is present.

The format for a conditional variable test is:

$$\text{IF } [\text{NOT}] \text{ condition-name}$$

If the condition-name is associated with a range or ranges of values (i.e., the "VALUES ARE" clause contains at least one "literal-1 THRU literal-2" phrase) then the condition represented by the condition-name is true if, for any of the associated range of values, the conditional variable is not less than literal-1 and not greater than literal-2.

4. Switch Status Tests

In the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION, the programmer may associate a condition-name with the "on" or "off" status of a software switch. (The switch is "on" when its value is 1, "off" when its value is zero.)

The status of such a switch may then be tested via a statement using the following format:

IF [NOT] condition-name

Results of this test are determined by the following table:

SPECIAL-NAMES STATEMENT	SWITCH STATUS	TEST	
		IF Condition-name	IF NOT Condition-name
...ON STATUS	off	False	True
IS condition-name	on	True	False
...OFF STATUS	off	True	False
IS condition-name	on	False	True

Compound Conditions

Simple conditions can be combined with logical operators according to specified rules to form compound conditions.

The most general form of a compound condition is:

$$\text{Compound-condition} = \text{simple-condition-1} \left\{ \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \right\} \text{simple-condition-2} \left\{ \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \right\} \text{simple-condition-n}$$

Here each simple condition can be either a relation test, a CLASS test, or a conditional variable test.

Parentheses may be used to indicate grouping as specified below. The following are illustration of compound conditions.

EXAMPLES:

AGE IS LESS THAN MAX-AGE AND AGE IS GREATER THAN 20

AGE IS GREATER THAN 25 OR MARRIED

STOCK-ON-HAND IS LESS THAN DEMAND OR STK-SUPPLY IS GREATER THAN DEMAND PLUS INVENTORY

A IS EQUAL TO B AND C IS NOT EQUAL TO D OR E IS UNEQUAL TO F AND G IS POSITIVE OR H IS LESS THAN I * J

STK-ACCT IS GREATER THAN 72 AND (STK-NUMBER IS LESS THAN 100 OR STK-NUMBER EQUALS 76290)

Note that it is not necessary to use the same logical connective throughout.

Letting A and B represent simple conditions, the following table defines the interpretation of AND, OR, and NOT in compound conditions:

A	B	Not A	A AND B	A OR B
True	True	False	True	True
False	True	True	False	True
True	False	False	False	True
False	False	True	False	False

Thus, if A is TRUE and B is FALSE, then the expression A AND B is FALSE, while the expression A OR B is TRUE.

A compound condition is a sequence of simple conditions in which one or two words AND and OR appear between the conditions. The words AND and OR are called logical connectives when used in this sense. The word NOT is used preceding a left parenthesis, or preceding a simple condition which does not itself contain a NOT.

The rules for determining the logical value (true or false) of a compound condition are as follows:

1. If AND is the only logical connective used, then the compound condition is true if and only if each of the simple conditions is true.
2. If OR is the only logical connective used, then the compound condition is true if and only if one or more of the simple conditions is true.
3. If both AND or OR appear, then there are two cases to consider, depending on whether or not parentheses are used.
 - a. Parentheses can be used to indicate grouping. They must always be paired, as in algebra, and the expressions within the parentheses will be evaluated first. The precedence of nested parenthetical expressions is the same as normal algebra. That is, the innermost parenthetical expressions are evaluated first.
 - b. If parentheses are not used, then the conditions are grouped first according to AND, proceeding from left to right, and then by OR, proceeding from left to right.
4. When NOT precedes a parenthesized condition, it reverses the logical value of the parenthesized condition; that is, NOT (condition) is true when (condition) is false. For example, NOT (A AND B) is true if either A or B is false. Thus, NOT (A AND B) is equivalent to NOT A OR B, and is true when A and B are not both true (i.e., when either is false or both are false). Similarly, NOT (A OR B) is equivalent to NOT A AND NOT B, and is true only when A and B are both false.

EXAMPLES:

1. To evaluate C1 AND (C2 OR NOT (C3 OR C4)), use the first part of rule 3 and successively reduce this by substituting as follows:

Let C5 equal "C3 OR C4" resulting in C1 AND (C2 OR NOT C5)

Let C6 equal "C2 OR NOT C5" resulting in C1 and C6

This can be evaluated by the table shown above.

2. To evaluate C1 OR C2 AND C3, use the second part of rule 3 and reduce this to C1 OR (C2 AND C3), which can now be reduced as in Example 1.
3. To evaluate C1 AND C2 OR NOT C3 AND C4, group first by AND from left to right, resulting in:

(C1 AND C2) OR (NOT C3 AND C4)

which now can be evaluated as in the first example.

4. To evaluate C1 AND C2 AND C3 OR C4 OR C5 AND C6 AND C7 OR C8, group from the left by AND to produce:

((C1 AND C2) AND C3) OR C4 OR ((C5 AND C6) AND C7) OR C8

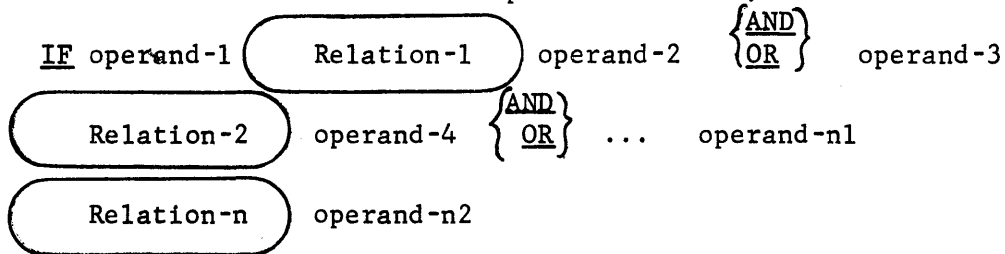
which can now be evaluated as in Example 1.

Abbreviations

Only conditions involving full relation tests have three terms (a subject, a relation, and an object). In order to simplify writing long expressions, COBOL allows the omission of some of these terms in certain forms of compound conditions. To simplify the narrative, the following representation is used in the remainder of the discussion in this section:

operand-i represents $\left\{ \begin{array}{l} \text{data-name-i} \\ \text{literal-i} \\ \text{formula-i} \end{array} \right\}$

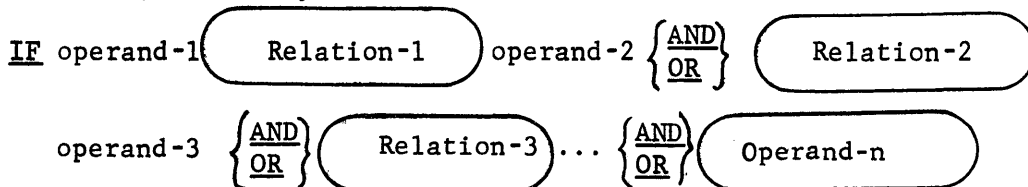
The general form of a compound condition containing only full relation tests (hereafter called a relational compound condition) is:



Of the five examples given earlier in this section, only the first, third, and fifth are of this form.

Abbreviation 1:

When a relational compound condition has the same term immediately preceding each relation, then only the first of these terms need be written. Thus:



is interpreted as if operand-1 had appeared immediately preceding each

Relation-n.

This form of abbreviation is applicable independently of the presence or absence of parentheses.

Referring back to the first example, the example could also be written as:

IF AGE IS LESS THAN MAX-AGE AND GREATER THAN 20

As another illustration of this abbreviation, note that

IF A EQUALS B OR EQUALS C AND IS GREATER THAN D

is an abbreviation for

IF A EQUALS B OR A EQUALS C AND A IS GREATER THAN D

which is equivalent to

IF A EQUALS B OR (A EQUALS C AND A IS GREATER THAN D)

Abbreviation 2:

The second type of abbreviation allowed is in the case where both the subject and the relation are common. In this case, then only the first occurrence of the subject and the relation are written. Thus:

IF operand-1 Relation operand-2 $\left\{ \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \right\}$ operand-3
 $\left\{ \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \right\}$... $\left\{ \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \right\}$ operand-n

is equivalent to

IF operand-1 Relation operand-2 $\left\{ \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \right\}$ operand-1 Relation
 operand-3 $\left\{ \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \right\}$... $\left\{ \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \right\}$ operand-1 Relation operand-n

This form of abbreviation is applicable independently of the presence or absence of parentheses.

As an illustration of this form, note that the expression

IF A = B OR C AND D

is equivalent to

IF A = B OR A = C AND A = D

which is in turn equivalent to:

IF A = B OR (A = C AND A = D).

The following examples show an abbreviated form followed by its equivalent expansion:

1. IF A = B AND C EXCEEDS D OR = B
IF A = B AND C EXCEEDS D OR C = B
2. IF A = B MOVE X TO Y; OTHERWISE IF GREATER THAN C THEN ADD M TO N.
IF A = B MOVE X TO Y; OTHERWISE IF A IS GREATER THAN C THEN ADD M TO N.
3. IF A = B MOVE X TO Y; OTHERWISE IF GREATER, MOVE M TO N ELSE
MOVE P TO Q.
IF A = B MOVE X TO Y; OTHERWISE IF A IS GREATER THAN B MOVE M TO N ELSE MOVE P TO Q.
4. IF A = B OR IS GREATER THAN B THEN MOVE C TO A; IF GREATER THAN B THEN ADD B TO A.
IF A = B OR A IS GREATER THAN B THEN MOVE C TO A; IF A IS GREATER THAN B THEN ADD B TO A.
5. IF A EXCEEDS B OR EQUALS B AND X EQUALS Y THEN IF GREATER THAN B MOVE C TO D.
IF A EXCEEDS B OR (A EQUALS B AND X EQUALS Y) THEN IF X IS GREATER THAN B MOVE C TO D.
6. IF A EQUALS B AND X IS GREATER THAN Y THEN ADD C TO D; IF EQUAL MOVE C TO D.
IF A EQUALS B AND X IS GREATER THAN Y THEN ADD C TO D; IF X EQUALS Y MOVE C TO D.
7. IF A EQUALS B AND (C OR D AND E IS GREATER THAN Y) OR LESS THAN Z MOVE M TO N.
IF (A EQUALS B AND (A EQUALS C OR (A EQUALS D AND E IS GREATER THAN Y))) OR E IS LESS THAN Z MOVE M TO N.

Table of Legal Symbol Pairs Involving Conditions and Logical Connectives

		SECOND SYMBOL					
		C	OR	AND	NOT	()
FIRST SYMBOL	C	-	P	P	-	-	P
	OR	P	-	-	P	P	-
	AND	P	-	-	P	P	-
	NOT	P*	-	-	-	P	-
	(P	-	-	P	P	-
)	-	P	P	-	-	P

"P" indicates that the pair is permissible, and "-" indicates that the pair is not permissible. Thus, the pair "OR NOT" is permissible, while the pair "NOT OR" is not permissible.

*Permissible only if the condition itself does not contain a NOT.

FORMING COMPOUND CONDITIONS

Conditional expressions may contain data-names, formulas, literals, arithmetic operators, relations of equality and relative magnitude and the logical operators NOT, AND and OR. Sub-expressions may be contained in parentheses as required.

If either a condition-name (such as MARRIED) or a relation (such as PAY IS GREATER THAN $2 * X - Y$) or a test is designated by the symbol C1, the following rules may be stated concerning the formation of compound conditions involving C1, NOT, AND and OR.

<u>The Condition</u>	<u>Is True If</u>
C1	C1 is true
Not C1	C1 is false
C1 and C2	Both C1 and C2 are true
C1 OR C2	Either C1 is true, C2 is true, or both are true
NOT (C1 AND C2)	"NOT C1 OR NOT C2" is true
NOT (C1 OR C2)	"NOT C1 AND NOT C2" is true

The conditional expression "C1 OR C2 AND C3" is identical with "C1 OR (C2 AND C3)" but is not the same as "(C1 OR C2) AND C3." In other words, compound conditions are grouped first according to AND and subsequently by OR. However, the programmer's use of parentheses will affect the order of grouping.

DECLARATIVES

Declaratives consist of compiler-directing sentences and the associated procedures. If present, declaratives must be grouped together at the beginning of the PROCEDURE DIVISION: the group of declaratives must be preceded by the key word DECLARATIVES, and must be followed by the key words, END DECLARATIVES.

COMPILER DIRECTING DECLARATIVES

A USE DECLARATIVE is used to specify procedures which are to supplement the standard procedures provided by the Input/Output system. The USE sentence immediately follows a section-name sentence, specifying the conditions under which the section is to be executed. Only PERFORM statements may reference all or part of a USE section. Within a USE procedure there must be no reference to the main body of the PROCEDURE DIVISION.

The format for the USE DECLARATIVE is:

section-name SECTION. USE -----.

Paragraph-name. First procedure statement...

Complete rules for writing the formats for USE are stated under the USE verb.

FORMULAS

A formula is an algebraic expression consisting of a combination of data-names and/or literals (representing items on which arithmetic may be performed) and arithmetic operators.

Basic Operators

There are five arithmetic operators which may be used in formulas. They may be expressed by characters (or symbols), in which case they must be surrounded by spaces, or by English equivalents. The following operators are available:

<u>Operation</u>	<u>Character</u>	<u>English Equivalent</u>
(Addition)	+	<u>PLUS</u>
(Subtraction)	-	<u>MINUS</u>
(Multiplication)	*	<u>MULTIPLIED BY</u> or <u>TIMES</u>
(Division)	/	<u>DIVIDED BY</u>
(Exponentiation)	**	<u>EXPONENTIATED BY</u>

The rules for forming algebraic expressions assume the existence of a precedence table for the arithmetic operators which, unless parenthesizing is used to modify the hierarchy, determines the sequence in which the arithmetic operations in a formula will be executed.

Understood precedence, from high to low, is:

Unary - (logical NOT)
 Exponentiation
 Multiplication and Division
 Addition and Subtraction

Parentheses may be used to override understood precedence. When the sequence of execution is not specified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right. Thus, expressions ordinarily considered to be ambiguous, such as $A/B*C$ and $A/B/C$, are permitted in COBOL. They are interpreted as if they were written $(A/B)*C$ and $(A/B)/C$, respectively.

A formula containing a double exponentiation (A^{B^C}), cannot be written in the form $(A**B**C)$; it must be written either $(A**(B**C))$ or $(A**B)**C$, whichever is intended.

An item or literal being exponentiated must not have a negative value. If data-item to be exponentiated can assume a negative value, a test (IF... NEGATIVE) should be arranged to bypass the exponentiation in case the value is negative.

FORMATION OF SYMBOL PAIRS

		SECOND SYMBOL				
		VARIABLE	*, /, **, +, -	Unary -	()
FIRST SYMBOL	VARIABLE	-	P	-	-	P
	*, /, **, +, -	P	-	P	P	-
	Unary -	P*	-	-	P	-
	(P	-	P	P	-
)	-	P	-	-	P

* Permissible only if the variable is not a literal.

A formula may never begin or end with an arithmetic operator except that a formula can begin with either of the symbols, +, -, or their English equivalents. In every formula there must be the same number of right parentheses as left parentheses.

VERBS

LISTED BY CATEGORIES

Arithmetic	{ ADD SUBTRACT MULTIPLY DIVIDE COMPUTE
Input-Output	{ ACCEPT READ WRITE OPEN CLOSE DISPLAY
Procedure Branching	{ GO ALTER PERFORM
Data Movement	{ MOVE EXAMINE
Ending	{ STOP
Compiler Directing Verbs	{ CALL COPY ENTER EXIT NOTE
Compiler Directing Declaratives	{ USE
Report Writer Verbs	{ GENERATE INITIATE TERMINATE
SORT Verbs	{ RELEASE RETURN SORT

SPECIFIC VERB FORMATS

The specific verb formats, together with a detailed discussion of the restrictions and limitations associated with each, appear on the following pages, in alphabetic sequence.

ACCEPT

FUNCTION: To receive low volume data from special sources.

ACCEPT data-name [FROM mnemonic-name]

Notes:

1. The ACCEPT verb may be used to obtain input data from any of the following sources:
 - GEIN (i.e., data cards supplied at object-time via a \$ DATA control card on which the file-code I* is punched). (See the GECOS manual.)
 - Switches (a special software feature provided by GECOS).
 - GETIME (a GECOS feature which emits the current date and time, in a standard format, upon request).
 - GELAPS (a GECOS feature which emits the current elapsed running time, in a standard format, upon request).
 - The console typewriter keyboard.

When the FROM option is not used, the input source is assumed to be GEIN. The FROM option must be specified for any other input source, and the mnemonic-name must then be a user-supplied word associated with an input data source by a statement in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION. Specific conventions for the various ACCEPT sources are given in notes 2 through 6 below.

2. If no mnemonic-name is specified, ACCEPT obtains data from GEIN. Data-name must then be a DISPLAY item of no more than 80 characters in size. The input item is assumed to occupy the leftmost character positions of the card. No automatic format check or conversion is provided, so it is recommended that the user employ IF tests to assure that the input card contents satisfy the description of data-name. Similarly no automatic end-of-file provision is available, so the user must provide his own test for end-of-file if the volume of GEIN data can vary. When the end-of-file condition occurs, ACCEPT obtains an implicit blank card; subsequently executed ACCEPT statements then continue to obtain blank cards. A test for ALL SPACES may thus suffice for end-of-file recognition, provided blank cards are not otherwise expected.
3. If a mnemonic-name associated with switches is specified, ACCEPT causes the value of data-name to be set to 1 if the switch is "on", or set to 0 (zero) if the switch is "off". Data-name must be a WORKING-STORAGE item whose description is equivalent to the following:

77 data-name; PICTURE 9; COMPUTATIONAL-1.

4. If a mnemonic-name associated with GETIME is specified, ACCEPT causes data-name to receive the current date and time values. Time resolution is in units of 1/64 millisecond. Data-name must be a WORKING-STORAGE item whose description is equivalent to the following:

```

01  data-name .
    02 MONTH; PICTURE 99.
    02 DAY; PICTURE 99.
    02 YEAR; PICTURE 99.
    02 TIME; PICTURE 9(8); COMPUTATIONAL-1.

```

5. If a mnemonic-name associated with GELAPS is specified, ACCEPT causes data-name to receive the elapsed running time charged thus far to the requesting program. Time resolution is in units of 1/64 millisecond. Data-name must be a WORKING-STORAGE item whose description is equivalent to the following:

```

77 data-name; PICTURE 9(8) ; COMPUTATIONAL-1.

```

6. If a mnemonic-name associated with the typewriter keyboard is specified, ACCEPT causes data-name to receive whatever value the operator types. Data-name must be a DISPLAY item of no more than 54 characters in size. No automatic format check is provided, so it is recommended that the user employ IF tests to assure that the type-in satisfies the description of data-name. A special provision for this use of ACCEPT is available via SPECIAL-NAMES and the DISPLAY verb. In SPECIAL-NAMES, the user may associate a mnemonic-name with the CONSOLE. A DISPLAY UPON (as it were) CONSOLE causes the displayed message to be saved until a subsequent ACCEPT references the console typewriter. The DISPLAY statement may appear anywhere in the program, as long as it is executed prior to the ACCEPT. An appropriate DISPLAY statement of this kind should always be used, to inform the operator what is expected of him.

Typewriter input/output is strongly discouraged except in extraordinary circumstances.

7. Note that a mnemonic-name does not have a data description in the Data Division but is defined only under SPECIAL-NAMES.

ADD

FUNCTION: To add two or more numeric data items and set the value of an item equal to the results.

Option 1:

ADD { literal-1
data-name-1 } [, { literal-2
data-name-2 }] { TO
GIVING }
data-name-n [ROUNDED] [; ON SIZE ERROR any imperative statement]

Option 2:

ADD { CORRESPONDING
CORR } data-name-1 TO data-name-2 [ROUNDED]

Notes:

1. In Option 1, the data-names used must refer only to elementary items. If GIVING is used, data-name-n is not used as an operand; hence, its format may contain editing symbols. In all other cases, the data-names used must refer to NUMERIC items only.
2. An error will be indicated at compilation time if the data description of any item used as an operand of any arithmetic verb (addends, subtrahends, multipliers, etc.) specifies the presence of editing symbols. Operational signs and implied decimal points are not considered editing symbols.
3. The maximum size of any operand (literal or data-name) is 18 decimal digits. An error will be indicated at compilation time if the data description of a data item used as an operand specifies a size in excess of 18 digits or a literal used as an operand contains more than 18 digits.
4. In Option 1, if no GIVING clause is used, whether or not the TO clause is used, the values of data-name-1 or literal-1 through data-name-n will be added together, and the sum will be stored as the value of data-name-n.
5. If the GIVING option is used, the sum of the values of the operands preceding the GIVING will be stored as the new value of data-name-n.
6. The Internal Formats of operands referred to in an ADD statement may differ among each other. Any necessary format transformation or decimal point alignment is automatically supplied throughout the calculation.
7. If the number of decimal places in a calculated result (sum) is greater than the number of decimal places associated with the resultant-data-name, truncation will occur, unless the ROUNDED option appears.

Truncation is always in accordance with the size associated with the resultant data-name. When the `ROUNDED` option is specified, it causes the least significant digit of the new value of the resultant-data-name to have its value increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

The following examples illustrate the effect of the `ROUNDED` option:

<u>Calculated Result</u>	<u>PICTURE Of Resultant Item</u>	<u>Value Stored</u>
3.14	·S9V9	3.1
3.15	S9V9	3.2
-3.14	S9V9	-3.1
-3.15	S9V9	-3.2

8. Whenever the magnitude of the calculated result exceeds the largest magnitude that can be contained in the resultant data-name, a size error condition arises. In the event of a size error condition, one of two possibilities will occur, depending on whether or not the `ON SIZE ERROR` option has been specified. The testing for the size error condition occurs only when the `ON SIZE ERROR` option has been specified.
 - a. In the event that `ON SIZE ERROR` is not specified and size error conditions arise, the value of the resultant-data-name may be unpredictable.
 - b. If the `ON SIZE ERROR` option has been specified and size error conditions arise, then the value of the resultant-data-name will not be altered. Instead, the imperative-statement associated with the `ON SIZE ERROR` option will be executed.
9. An `ADD` statement must refer to at least two addends.
10. Option 2 leads to several separate addition operations. Rules for the `CORRESPONDING` option for `ADD` are equivalent to those for `MOVE`, except that only numeric elementary items may correspond.
11. `CORR` is an abbreviation for `CORRESPONDING`.

ALTER

FUNCTION: To modify the destination of a GO statement.

ALTER procedure-name-1 TO PROCEED TO procedure-name-2

[procedure-name-3 TO PROCEED TO procedure-name-4...]

Notes:

Procedure-name-1, procedure-name-3, ..., are names of paragraphs which each contain a single sentence consisting of only a GO statement as defined under Option 1 of the GO verb.

CALL

FUNCTION: To permit transfer of control to a separately compiled subprogram or entry point within a subprogram, with a standard return mechanism provided. (The generated coding employs the CALL macro of GMAP.)

```
CALL routine-name [ USING data-name-1
                   [, data-name-2 . . .] ]
```

Notes:

The following conventions govern the use of CALL.

1. If the subprogram being called is an independently compiled COBOL program, routine-name must be its PROGRAM-ID.
2. If the routine-name being called is an entry-name (explicit) in an independently compiled COBOL program, the entry-name must be one specified somewhere in the called subprogram using the ENTRY POINT statement.
3. If the subprogram being called has been developed via another language, routine-name must be the subprogram's identification or entry-name according to the pertinent language's rules.
4. If routine-name is a paragraph-name or section-name within the current source program, CALL has exactly the same effect as "PERFORM routine-name" (without THRU or any other PERFORM option).
5. The implied entry point of a subprogram written in COBOL is the first non-DECLARATIVE procedural statement. This entry point is produced automatically by the compiler. The implied exit point follows immediately after the last procedure statement in the source program and is also produced automatically. It is the effective exit point when a subprogram is called by its PROGRAM-ID.

6. Any object-program produced by the COBOL compiler can be called as a subprogram from other object-programs. COBOL object-programs may also have multiple entry points which can be called and executed from other object-programs. Normally, such subprograms should not contain STOP RUN. The subprogram or entry point procedures should be written to allow control to eventually pass to an appropriate exit point. An EXIT paragraph must be used if a subprogram has alternative routes to the exit point. At least one EXIT entry-name must be specified for each ENTRY POINT statement in a subprogram.
7. A called subprogram may CALL other COBOL subprograms by implied entry point or by entry-names specified within the other subprograms. No subprogram may CALL its own PROGRAM-ID nor entry-names within itself. Entry point procedures may CALL entry-names in other subprograms but extreme care should be used to avoid a CALL into a subprogram which has any previous active CALL still current.
8. The USING clause is utilized to specify an argument list for the subprogram being called. The USING arguments are not valid when a CALL references a COBOL subprogram by its PROGRAM-ID. They are valid when referencing explicit entry points within a COBOL subprogram. The user must assure that the order and descriptions of the arguments conform to the called subprogram's requirements. At most, ten arguments may be specified.
9. The USING clause specifies "input" and "output" arguments to the called subprogram. USING data-names must reference Working-Storage or Constant Section items or items in files for which a PROCESS AREA is specified. A USING argument may be a file-name, in which case the object-program argument will be a File Control Block pointer. File-names are not valid arguments when calling a COBOL subprogram; they are restricted to called subprograms developed in another language.
10. USING data-names must not be subscripted.
11. The user must assure that the number of USING arguments specified in a CALL to an ENTRY POINT corresponds exactly with the number of USING and GIVING arguments specified for its ENTRY POINT and that the data descriptions for each pair of the corresponding arguments are identical.

CLOSE

FUNCTION: To terminate the processing of input and output reels and files.

Option 1:

CLOSE file-name-1 [WITH { NO REWIND / LOCK }] [, file-name-2 ...]

Option 2:

CLOSE file-name-1 [REEL [WITH { NO REWIND / LOCK }]] .

Notes:

1. For the purpose of showing the effects of the various CLOSE options, all input and output files are divided into the following categories:
 - a. Non-tape; a file whose input or output medium is such that the concepts of rewinding and reels have no meaning.
 - b. Partial-reel; a file which is entirely contained on a reel requiring the MULTIPLE FILE TAPE CONTAINS clause in the I-O-CONTROL paragraph of the ENVIRONMENT DIVISION.
 - c. Single-reel; a file which is entirely contained on one reel and is the only file on that reel.
 - d. Multiple-reel; a file which may be contained on more than one reel, and the number of reels might possibly be greater than the number of tape units assigned.
2. The results of executing each CLOSE option for each type of file are summarized in the following table. The definitions of symbols in the table immediately follow the table. Where the definition depends on whether the file is an input or an output file, alternate definitions are given; otherwise, the one definition applies to both input and output files.

File Type \ CLOSE Option	non-tape	partial-reel	single-reel	multiple-reel
CLOSE	F	F,W	F,W	F,W,N
CLOSE WITH LOCK	F,L	F,W,L	F,W,L	F,W,L,N
CLOSE WITH NO REWIND	X	F,C	F,C	F,C,N
CLOSE REEL	X	X	X	R,K

- (C) (No rewind of current reel.) The current reel is left in its current position.
- (F) (Standard CLOSE file.)

Input files: If the file is positioned at the end of the file, and there is an ending label record, the ending label record is checked, the data area is released, and the standard closing conventions are performed. If the file is positioned at the end of the file, and there is no ending label record, the data area is released and the standard closing conventions are performed, but no ending label checking is performed. An input file is considered to be positioned at the end of the file if an explicit or implicit AT END exit has been executed and no CLOSE statement has been executed.

Output files: If an ending label record has been described for the file, it is constructed and written on the output medium, the data area is released, and the standard closing conventions are performed. If no ending label record has been described for the file, the data area is released and the standard closing conventions are performed.

- (L) (Standard file lock.) An appropriate technique is supplied to insure that this file cannot be OPENed again in this object program. If an attempt is made to OPEN the file, an error indication will occur.
- (K) (Standard Reel Lock.) This current reel is rewound with hardware interlock.
- (N) (Previous reels unaffected)

Input files: All reels in the file prior to the current reel are processed according to the standard tape swap procedure regardless of the CLOSE file option, except those reels controlled by a prior CLOSE REEL statement. If the current reel is not the last reel in the file, the reels in the file following the current reel are not processed in any way.

Output files: All reels in the file prior to the current reel are processed according to the standard tape swap procedure regardless of the CLOSE file option, except those reels controlled by a prior CLOSE REEL statement.

- (R) (Standard CLOSE REEL.)

Input Files: The standard end of reel operations described under READ are performed, except that no ending label checking or any USE procedures related to ending label checking is performed.

Output files: The standard end of reel operations described under WRITE are performed.

- (W) (Rewind) Rewind the current reel.
 - (X) (Illegal) This is an illegal combination of a CLOSE option and a file type and the results at Object Time are unpredictable.
3. All files which have been OPENed must be CLOSEd prior to the execution of a STOP RUN statement. No automatic CLOSEing takes place.
 4. If the file has been specified as OPTIONAL (see the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION), the standard end of file processing is not performed whenever this file is not present.

COMPUTE

Function: To assign to a data item the value of a numeric data item, literal, or formula.

COMPUTE data-name-1 [ROUNDED]

{ FROM } { data-name-2 }
 { = } { literal-1 } [; ON SIZE ERROR any imperative
 { EQUALS } { formula } statement]

Notes:

1. Literal-1 must be a numeric literal. The data-names used must refer only to elementary items. Data-name-1 is not used as an operand; hence, its format may contain editing symbols. In all other cases, the data-names used must refer to NUMERIC items only.
2. The data-name-2 and literal-1 options provide an alternative method in COBOL for setting the value of data-name-1 equal to the value of data-name-2 or literal-1.
3. The formula option permits the use of any meaningful combination of data-names (which must satisfy the general rules specified for data-names used in the simple arithmetics), numeric literals, and arithmetic operators, parenthesized as required.
4. The ON SIZE ERROR option applies only to the final result and does not apply to any of the intermediate results.
5. All rules regarding the ON SIZE ERROR, the ROUNDED option, the size of operands, truncation and the editing of results, which are specified for the simple arithmetics, apply also to the COMPUTE verb.
6. The words FROM and EQUALS are equivalent to each other and to the symbol"=". They may be used interchangeably and the choice is generally made for readability.

COPY

FUNCTION: The COPY statement incorporates procedures from a library into the source program.

paragraph-name. COPY library-name.

Notes:

1. The duplication process replaces the COPY statement by the procedure statements that follow the paragraph-name specified as the library-name. The duplication process will insert all procedural statements following the library-name, up to but excluding the next paragraph-name.
2. No other statement or clause may appear in the same entry as the COPY statement.
3. When the COPY statement is used, the COPY and the statements associated with library-name (paragraph to be copied) will appear on the output listing.
4. The COPY statement should not be used to duplicate section-names.
5. All paragraph-names associated with a COPY statement must begin in column 8 of the source card.
6. The text contained on the library must not contain any COPY statements.
7. Information associated with building a library file can be found in Appendix Q.

4. If a mnemonic-name associated with the TYPEWRITER is specified, DISPLAY causes a single line to be typed, with no keyboard response expected. A typewriter line must not exceed 72 character positions. Additional rules are given in Note 6. Typewriter output is strongly discouraged except in extraordinary circumstances.
5. If a mnemonic-name associated with the CONSOLE is specified, DISPLAY causes a single line to be readied for typing, but not immediately typed. Instead, the line is saved until a subsequent ACCEPT statement is executed, at which time the line is typed. This use of DISPLAY is specifically designed to supplement ACCEPT, so that an explanatory timeout can immediately precede a keyboard input request. The ACCEPT and DISPLAY statements need not appear together in the source-program, provided the DISPLAY is executed first. Should no ACCEPT statement be executed after the DISPLAY, the output line is not typed. Two DISPLAY statements of this kind with no intervening ACCEPT would result in suppression of the first line. A typewriter line must not exceed 72 character positions. Additional rules are given in Note 6.
6. The following rules apply to all DISPLAY statements transmitting data to SYSOUT or the typewriter:
 - a. When the DISPLAY consists of multiple operands, the data comprising the first operand is displayed as the first set of characters, the data comprising the second operand as the second set, etc. Operands are not automatically separated by spaces.
 - b. DISPLAY data begins in the first character position of a line, subject to the effects of horizontal and vertical tabulation control characters embedded in the data. If the user employs such characters, he must assure that the line-length limits given above are not exceeded.
 - c. Data-name-1, data-name-2, etc. must have DISPLAY or DISPLAY-1 USAGEs. The literals may be figurative constants, in which case their size is understood to be one character. ALL has no significance here.
7. The use of DISPLAY statements transmitting data to SYSOUT in a multi-segment environment (see Chapter X) must be carefully planned to avoid possible overlay of the COBOL subroutine ".COSYS" which controls output for SYSOUT displays. One way is to use DISPLAY in a main segment which will never be overlaid, whether or not its actual execution in such a segment is desired.

DIVIDE

FUNCTION: To divide one numerical data-item into another and set the value of an item equal to the result.

DIVIDE { data-name-1 } INTO
 { literal-1 }
 { data-name-2 [GIVING data-name-3] } [ROUNDED]
 { literal-2 GIVING data-name-3 }
 [; ON SIZE ERROR any imperative statement]

Notes:

1. The data-names used must refer only to elementary items. If GIVING is used, data-name-3 is not used as an operand; hence, its format may contain editing symbols. In all other cases, the data-names used must refer to NUMERIC items only.
2. All rules specified under the ADD verb regarding the size of operands, presence of editing symbols in operands, the ON SIZE ERROR option, the ROUNDED option, the GIVING option, truncation, and the editing of results, apply to the DIVIDE verb.
3. When GIVING is not used, the initial value of data-name-1 or literal-1 will be divided into the initial value of data-name-2. The value of the dividend will be replaced by the value of the quotient resulting from the operation. Note that a literal must not be used as a dividend.
4. Division by zero constitutes a special type of "size error". Program control may be provided through the use of a test for zero prior to attempting division. If the "zero test" type of program control is not provided, the rules specified under the ADD verb with respect to the ON SIZE ERROR option apply.

ENTER

FUNCTION: To permit the inclusion in the source-program of statements which are not defined in the COBOL language.

Option 1:

ENTER { TIME-SAVING
SPACE-SAVING } MODE.

Option 2:

[paragraph-name.] ENTER GMAP.

⋮ } GMAP coding

ENTER COBOL.

Option 3:

[paragraph-name.] ENTER DEFINITIONS.

SYMBOL literal-1 { EQUALS } { [SIZE OF
=] [INITIAL CHARACTER OF] data-name-1 }
PROCEDURE procedure-name-1

[SYMBOL literal-2 { EQUALS } { [INITIAL CHARACTER OF] data-name-2 } . . .].
=] PROCEDURE procedure-name-2

ENTER COBOL.

Option 4:

[paragraph-name.] ENTER LINKAGE MODE.

[CALL routine-name [USING data-name-1 [, data-name-2 . . .]]]

[ENTRY POINT IS entry-name [[USING data-name-1 [, data-name-2 . . .]]
[GIVING data-name-3 [, data-name-4 . . .]]]]]

[POPUP procedure-name]

[{ CALL
ENTRY . . . }].
POPUP

ENTER COBOL.

The Notes are given separately for each of the above Options. Although many COBOL compilers permit use of ENTER, the user should be aware that ENTER specifications generally differ completely from one computer line to another. ENTER statements in a source-program must consequently be changed before the program can be compiled on a computer line other than that for which the source-program was originally prepared.

ENTER { TIME-SAVING
SPACE-SAVING } MODE.

Notes:

1. Under some circumstances, the compiler is capable of producing two alternative sets of object-program instructions for a given procedural statement, the **TIME-SAVING** version requiring more space but less execution time than the **SPACE-SAVING** version. (Generally speaking, such alternatives are available in the compiler only when the trade-off exceeds two-for-one in both categories.) This **ENTER** verb option permits the user to take advantage of such possible alternatives on the basis of his knowledge of the relative level of activity of various parts of the program.
2. The normal mode is **TIME-SAVING**. The **SPACE-SAVING** mode applies to all statements following **ENTER SPACE-SAVING** until **ENTER TIME-SAVING** is encountered, at which point the compiler resumes the normal **TIME-SAVING** mode.
3. A given set of procedural statements does not necessarily entail any for which the **TIME-SAVING** or **SPACE-SAVING** mode options are significant. In such procedures, the mode **ENTERed** has no effect on the object-program.

[paragraph-name.] ENTER GMAP.

. } GMAP coding
 . }
 . }
 . }
ENTER COBOL.

Notes:

1. The GMAP coding following an ENTER GMAP statement must be terminated by an ENTER COBOL statement which begins on a new line. The ENTER COBOL statement cannot have a paragraph-name. The lines intervening between the ENTER GMAP and ENTER COBOL statements must consist of GMAP coding.
2. A special format is used for the GMAP statements; it is in effect the standard GMAP format shifted six places to the right.

<u>Columns</u>	<u>Interpretation</u>
1-6	COBOL sequence number
7-12	Location field
13	Even/odd subfield
14-19	Operation field
20-21	Blank
22-72	Variable field
73-80	Program identification

Information to the right of column 72 is not interpreted as part of the variable field. As in ordinary GMAP coding, comments must be separated from variable field information by at least one blank.

3. GMAP symbols defined in the location field must not conflict with reserved system symbols (consult the GMAP manual for a listing of reserved symbols). They must follow the GMAP rules for symbol formulation. Symbols reserved for compiler use which must not be defined in the location field of GMAP statements include:
 - a. Symbols in the form LNNNNN, where "L" is any letter and "NNNNN" is any string of 5 digits.
 - b. Symbols having as their left-most two characters any file-code specified in the Environment Division.
 - c. Symbols of the form VfEOF, where "fc" is any file-code specified in the Environment Division.
 - d. The PROGRAM-ID defined in the Identification Division.
 - e. The symbol ENTER.

- f) Symbols having as their left-most two characters either ".C" or "C."
 - g) Any 6-character symbol with the first character numeric.
4. COBOL data-names and procedure-names are not directly accessible to GMAP coding; a special provision (see ENTER DEFINITIONS) permits GMAP symbols to be applied to COBOL procedures and data items. The "locsymb" of each GEFRC File Control Block (see the GEFRC manual) is the two-character file-code assigned to the file in a SELECT sentence in the ENVIRONMENT DIVISION followed by the characters "FICB ". For a file with explicit or implicit PROCESS AREA, the beginning location of the PROCESS AREA has the symbol fcRECD where "fc" is the file-code assigned. For any other file, a "current record location" pointer is available (provided the file is OPEN) in the address field (bits 0-17) of the File Control Block word with symbol "fc" (file-code assigned). Data must be addressed relatively to the current record location pointer for such a file.
5. All GMAP rules must be observed in the GMAP coding. Use of pseudo-operations which alter the location counter is discouraged; unless greater care is taken, such pseudo-operations can lead to unpredictable results at object-time. Their use should presuppose a thorough understanding by the programmer of the location counter conventions followed by the COBOL compiler in generated coding.

ENTER DEFINITIONS.

$$\underline{\text{SYMBOL}} \text{ literal-1} \left\{ \begin{array}{c} \underline{\text{EQUALS}} \\ = \end{array} \right\} \left\{ \begin{array}{l} [\underline{\text{SIZE OF}}] \\ [\underline{\text{INITIAL CHARACTER OF}}] \text{ data-name} \\ \underline{\text{PROCEDURE}} \text{ procedure-name} \end{array} \right\}$$
[; SYMBOL]ENTER COBOL.

Notes:

1. Literal-1 must be a non-numeric literal, consisting of one to six characters which satisfy the GMAP rules for symbol formation.
2. Data-name must be defined in the Data Division in a Record Description entry. It may be qualified, but not subscripted. Procedure-name must be defined in the Procedure Division as a paragraph-name or section-name. It may be qualified (if it is a paragraph-name). A COBOL rule prohibits use of the same name for both data and procedures.
3. A SYMBOL statement which references a data-name or a procedure-name only causes the compiler to equate the given GMAP symbol (literal-1) to the first memory location of the specified data area or procedural instructions.
4. A SYMBOL statement which references a data-name using the INITIAL option causes the compiler to equate the symbol to an appropriate number (0, 1, ..., 5) indicating the first character position occupied by the data item within the (first) computer word. The left-most position is indicated by 0, the right-most by 5.
5. A SYMBOL statement which references a data-name using the SIZE option causes the compiler to equate the symbol to an appropriate number which indicates the number of BCD character positions required to contain the data-name.

ENTER LINKAGE MODE.

{	<u>CALL</u> routine-name [<u>USING</u> data-name-1 [, data-name-2 . . .]]	}
	<u>ENTRY POINT IS</u> entry-name	
	[[<u>USING</u> data-name-1 [, data-name-2 . . .]]	
	[<u>GIVING</u> data-name-3 [, data-name-4 . . .]]]	
	<u>POPUP</u> procedure-name	

ENTER COBOL.

Notes:

1. CALL permits transfer of control to a separately compiled subprogram or entry point within a subprogram with a standard return mechanism provided. (The generated coding employs the CALL macro of GMAP.) The following conventions govern the use of CALL.
 - a) If the subprogram being called is an independently compiled COBOL program, routine-name must be its PROGRAM-ID.
 - b) If the routine-name being called is an explicit entry-name in an independently compiled COBOL subprogram, the entry-name must be one specified in an ENTRY POINT statement in the independent subprogram.
 - c) If the subprogram being called has been developed via another language, routine-name must be the subprogram's identification or entry-name according to the pertinent language's rules.
 - d) If routine-name is a paragraph-name or section-name within the current source-program, CALL has exactly the same effect as "PERFORM routine-name" (without THRU or any other PERFORM option).
 - e) The implied entry point (PROGRAM-ID) of a subprogram written in COBOL is the first non-DECLARATIVE procedural statement. This entry point is produced automatically by the compiler. The implied exit point follows immediately after the last procedural statement in the source program and is also produced automatically. It is the effective exit point when a subprogram is called by its PROGRAM-ID.

- f) Any object-program produced by the COBOL compiler can be called as a subprogram from other object-programs. COBOL object-programs may also have multiple entry points which can be called and executed from other object-programs. Normally, such subprograms should not contain STOP RUN. The subprogram or entry point procedures should be written to allow control to eventually pass to an appropriate exit point. An EXIT paragraph must be used if a subprogram has alternative routes to the exit point. At least one EXIT entry-name must be specified for each explicit entry-name in a subprogram.
- g) A called subprogram may CALL other COBOL subprograms by implied entry point or by entry-names specified within the other sub-programs. No subprogram may CALL its own PROGRAM-ID nor entry-names within itself. Entry point procedures may CALL entry-names in other subprograms; however, extreme care should be used to avoid a CALL into a subprogram which has any previous active CALL still current.
- h) The USING clause is utilized to specify an argument list for the subprogram being called. The USING arguments are not valid when a CALL references a COBOL subprogram by its PROGRAM-ID. They are valid when referencing explicit entry points within a COBOL subprogram. The user must assure that the order and descriptions of the arguments conform to the called subprogram's requirements. At most, ten arguments may be specified.
- i) The USING clause specifies "input" and "output" arguments to the called subprogram. USING data-names must reference Working-Storage or Constant Section items, or items in files for which a PROCESS AREA is specified. A USING argument may be a file-name, in which case the object-program argument will be a File Control Block pointer. File-names are not valid arguments when calling a COBOL subprogram; they are restricted to called sub-programs developed in another language.
- j) USING data-names must not be subscripted.
- k) The user must assure that the number of USING arguments specified in a CALL to an explicit entry-name corresponds exactly with the number of USING and GIVING arguments specified in its ENTRY POINT statement and that the data descriptions for each corresponding argument are identical.

2. ENTRY POINT provides a way to define entry points into a subprogram other than the implied entry point (the PROGRAM-ID) which is the first non-DECLARATIVE procedural statement.

The following conventions govern the use of ENTRY POINT.

- a) The entry-name specified must not contain more than six characters, with at least the first two characters a letter, and the remaining characters letters, and/or digits. It must not be the same as the PROGRAM-ID.
- b) An ENTRY POINT statement can be used any place in the Procedure Division except under DECLARATIVES.
- c) There must be no path of program flow to an ENTRY POINT statement within the program containing the ENTRY POINT statement. Hence, the statement should not have a paragraph-name.
- d) An ENTRY POINT statement can only be referenced by calls from external subprograms. It must not be referenced by a CALL from within the subprogram in which it resides.
- e) The user may specify as many ENTRY POINTS as desired for a particular subprogram.
- f) For each ENTRY POINT statement there must be at least one EXIT entry-name statement to provide the exit point for any CALL which references the entry-name.
- g) The USING clause is used to specify an input argument list for an ENTRY POINT. The USING data-names specified must be level 77 or 01 items specified in the Working-Storage or Constant Section. When a CALL references an ENTRY POINT with USING arguments, the compiler generates word moves for each argument, using the indirect address specified in the CALL as a sending field and the corresponding data-name from the USING argument list of the ENTRY POINT as a receiving field. The user must assure that the descriptions of the corresponding arguments are identical. Procedural statements following a particular ENTRY POINT will not be executed until all USING argument moves are complete.

- h) The GIVING clause is used to specify an output argument list for an ENTRY POINT. The GIVING data-names specified must be level 77 or 01 items specified in the Working-Storage or Constant Section. When a CALL references an ENTRY POINT with GIVING arguments, the compiler generates word moves for each argument, which will be executed on an EXIT entry-name for the referenced ENTRY POINT.

The GIVING moves use the corresponding indirect address specified in the CALL as a receiving field and the corresponding data-name from the GIVING argument list for the ENTRY POINT as a sending field.

- i) The user must assure a one-to-one correspondence between the CALL USING arguments and the ENTRY POINT USING/GIVING arguments. The data formats for the corresponding arguments must be identical.
- j) USING/GIVING data-names must not be subscripted.
- k) An entry-name must not be used in a \$ ENTRY control card.

3. POPUP permits extended use of PERFORM statements. Each PERFORM executed causes an exit link to enter a push-down stack which services all PERFORMs. The stack is so arranged that only the last PERFORM executed can engage its exit mechanism, if control passes to any other exit mechanism; even if a relevant PERFORM is active, the exit mechanism simply passes control to the next statement in the written sequence. When the exit mechanism of the last executed PERFORM is reached, it automatically removes its exit link and pops up the stack before returning control. This stack structure permits recursive PERFORMs, nested PERFORMs with crossing ranges, and multiple active PERFORMs with a common exit point. (Use of such extended capabilities, of course, requires that the user supply his own MOVES as necessary to avoid unintended overlaying of data with new values.) A push-down stack approach inherently requires that all pop-ups must eventually occur in an orderly way. This means that control must eventually reach the exit mechanism of any active PERFORM; and for nested PERFORMs control must pass to the exit mechanisms in the logical order "innermost to outermost." POPUP permits this inherent problem to be overcome. When the programmer plans the control sequence in such a way that not all exit mechanisms will be operated in the required order, he must employ POPUP to remove the unused exit links from the stack. The simplest example of such a situation is a GO transferring control outside of a PERFORM range without a subsequent return to the range. The following rules apply to POPUP:
 - a) Procedure-name must be the name of the last paragraph or section in the range of a currently active PERFORM. This rule is extremely important.
 - b) POPUP removes from the stack all exit links up to and including that associated with the exit mechanism following the "procedure name" paragraph or section. POPUP does not return control for any exit link; it merely passes control to the statement following it in the written sequence after its action on the stack.
 - c) If control permanently leaves a PERFORM range, the memory area allocated for the stack will gradually fill to overflow as the PERFORM is repeatedly executed unless POPUP is used. Stack overflow derails the object-program.
 - d) If an active PERFORM's exit mechanism is deliberately bypassed, prior PERFORMs which are still active cannot have their exit mechanisms return control unless POPUP is used.
 - e) POPUP must not reference an entry-name specified by the ENTRY POINT statement.

EXAMINE

FUNCTION: To replace and/or count the number of occurrences of a given character in a data item.

Option 1:

EXAMINE data-name TALLYING { UNTIL FIRST
ALL
LEADING } literal-1
[REPLACING BY literal-2]

Option 2:

EXAMINE data-name REPLACING { ALL
LEADING
[UNTIL] FIRST } literal-3 BY literal-4

Notes:

1. The description, appearing in the DATA DIVISION, of the data-name used in an EXAMINE statement must be such that USAGE is DISPLAY (explicitly or implicitly). Each literal used in an EXAMINE statement must consist of a single character belonging to a CLASS consistent with that of data-name.
2. Examination proceeds as follows:
 - a) For items whose CLASS is not NUMERIC, examination starts at the left-most character and proceeds to the right. Each character in the item specified by the data-name is examined in turn. Any reference to the first character means the left-most one.
 - b) If an item referenced by the EXAMINE verb is NUMERIC, it must consist of numeric characters and may possess an operational sign. Examination starts at the left-most character (excluding the sign) and proceeds to the right. Each character is examined in turn. Regardless of where the sign is physically located, it is completely ignored by the EXAMINE verb. Any reference to the first character means the left-most numeric character.
3. The TALLYING option creates an integral count which replaces the value of a special register called TALLY. The count represents the number of:
 - a. Occurrences of literal-1 when the ALL option is used.
 - b. Occurrences of literal-1 prior to encountering a character other than literal-1 when the LEADING option is used.
 - c. Characters not equal to literal-1 encountered before the first occurrence of literal-1 when the UNTIL FIRST option is used.

4. When either of the REPLACING options are used (i.e., with or without TALLYING) the replacement rules are as follows, subject to the rules in Note 2:
- a) When the ALL option is used, then literal-2 (or literal-4) is substituted for each occurrence of literal-1 (or literal-3).
 - b) When the LEADING option is used, the substitution of literal-2 (or literal-4) terminates as soon as a character other than literal-1 (or literal-3) or the right-hand boundary of the data item is encountered.
 - c) When the UNTIL FIRST option is used, the substitution of literal-2 (or literal-4) terminates as soon as literal-1 (or literal-3) or the right-hand boundary of the data item is encountered.
 - d) When the FIRST option is used, the first occurrence of literal-3 is replaced by literal-4.

4. If control reaches an EXIT statement without the PROGRAM or entry-name option and no associated PERFORM, SORT, or USE statement is active or if control reaches an EXIT PROGRAM statement and no CALL statement is active, control falls thru the EXIT point to the first sentence of the next paragraph.
5. If control reaches an EXIT PROGRAM statement while the subprogram is being executed as the result of a CALL statement, control returns to the point in the calling program immediately following the CALL statement.
6. The EXIT PROGRAM or entry-name statement must not be used in the DECLARATIVES.
7. The EXIT entry-name option defines an exit point for a specific entry-name which is given in the associated ENTRY POINT statement. The entry-name must be defined by the ENTRY POINT statement in the same subprogram as the EXIT entry-name.
8. The user can specify more than one exit point for any given ENTRY POINT, but there must be at least one EXIT entry-name for each ENTRY POINT specified.
9. If control reaches an EXIT entry-name statement, a check is made to determine if the entry point corresponds to the current exit point. If they correspond, it provides an exit whereby the GIVING fields specified for the ENTRY POINT are moved, all registers restored, and control returns to the point immediately following the CALL statement for the given entry-name. If the EXIT does not correspond to the entry-name, control will pass through the EXIT entry-name statement to the first sentence of the next paragraph.
10. The user must insure when using an EXIT PROGRAM statement that no PERFORM is currently active in any subprogram at the time the EXIT PROGRAM statement may be encountered.

GENERATE

FUNCTION: To present a report entry based on PROCEDURE DIVISION control.

GENERATE data-name-1

Notes:

1. If data-name-1 represents the name of a TYPE DETAIL report group, GENERATE does all the relevant automatic operations and produces an actual output DETAIL report group. (Detail reporting.)
2. If data-name-1 is the name of an RD entry, GENERATE does all the relevant automatic operations and updates the FOOTING report group(s) within the report without producing an actual DETAIL report group associated with the report. Thus it increments (decrements) all SUM counters associated with the report description. (Summary reporting.) If the report includes more than one TYPE DETAIL report group, all SUM counters are incremented each time such a GENERATE is executed.
3. GENERATE, produces the following automatic operations (as needed):
 - a) Steps and tests the LINE-COUNTER and/or PAGE-COUNTER to produce appropriate PAGE or OVERFLOW FOOTING and/or PAGE or OVERFLOW HEADING report groups.
 - b) Recognizes any specified CONTROL breaks to produce appropriate CONTROL FOOTING and/or CONTROL HEADING report groups.
 - c) Accumulates into the SUM counters all specified data-name(s). Resets the SUM counters on an associated CONTROL break. Performs an updating procedure between CONTROL break levels for each set of SUM counters.
 - d) Executes any specified routines defined by USE before generation of the associated report group(s).
4. During the execution of the first GENERATE statement referring to a report or to a DETAIL report group within a report, all CONTROL HEADING report groups specified for the report are produced in the order major...minor, immediately followed by any DETAIL report group specified in the statement. If a data-name control break is recognized at the time of execution of a GENERATE statement (other than the first that is executed for a report), all CONTROL FOOTING report groups specified for the report are produced from the minor report group up to and including the report group specified for the data-name which

caused the control break. The CONTROL HEADING report group(s) specified for the report, from the report-group specified for the data-name which caused the control break down to the minor report group, are then produced in that order. The DETAIL report group specified in the GENERATE statement is then produced.

5. When data is moved to a report group it is edited according to the rules described under the MOVE verb.

GO

FUNCTION: To depart from the normal sequence of procedures.

Option 1:

GO TO [procedure-name-1]

Option 2:

GO TO procedure-name-1, procedure-name-2,[procedure-name-3...]

DEPENDING ON data-name

Notes:

1. If a GO statement is to be ALTERed, then:
 - a) The GO statement must itself have a paragraph-name, and must have the option 1 format.
 - b) The paragraph in which the GO statement is included must consist solely of the GO statement.

If procedure-name-1 is omitted, an error stop will occur at object time unless the GO statement is ALTERed before its first execution.

2. In option 2, the data-name must have a positive integral value. The branch will be to the 1st, 2nd, ..., nth procedure-name, as the value of data-name is 1, 2, ..., n. If the value of data-name is anything other than the integers 1, 2, ..., n then no transfer is executed and control passes to the next statement in the normal sequence for execution.

INITIATE

FUNCTION: To INITIATE the presentation of a report.

INITIATE { data-name-1 [data-name-2. . .]
ALL }

Notes:

1. INITIATE sets to zero all entries associated with this report which contain SUM clauses; the controls for all the TYPE report groups which are associated with this report are set up in their respective order.
2. The PAGE-COUNTER, if specified, is initially set to 1. If a different starting value for PAGE-COUNTER other than 1 is desired, the programmer may reset the COUNTER.
3. The LINE-COUNTER, if specified, is automatically set to zero before the INITIATE process.
4. The data-names must be defined by RD entries in the Report Section of the program.
5. A report can be reinitiated after it has been terminated.
6. If ALL is specified, all reports defined in the Report Section of the Data Division are initiated.
7. INITIATE does not open the file with which the report is associated. An OPEN statement for the file must be executed before the INITIATE statement.

MOVE

FUNCTION: To transfer data to one or more data areas, in accordance with the rules of editing.

MOVE { [[CORRESPONDING
CORR]] data-name-1 } TO data-name-2 [, data-name-3..
literal

Notes:

1. Additional receiving areas may be given following data-name-2. The data designated by the literal or data-name-1 will be moved first to data-name-2, then to data-name-3, etc. The notes referencing data-name-2 also apply to the other receiving areas.
2. Notes 2 through 4 refer to a MOVE without the CORRESPONDING option.
 - a) Any MOVE in which the sending and receiving items are both elementary items is an elementary MOVE.

In the following discussion PICTURE is used for clarity; however, every elementary item belongs to one of the categories listed below whether or not the PICTURE is used in its description:

N - Numeric - The item has CLASS NUMERIC. This includes any item whose PICTURE consists solely of characters from the set 9S V P and 0.

NE - Numeric Edited - The item has at least one of the following:

- (1) An editing clause (e.g., FLOAT DOLLAR SIGN, BLANK WHEN ZERO).
- (2) A PICTURE containing any of the numeric editing characters Z * \$, . + - CR and DB.
- (3) A PICTURE containing at least one of the insertion character B, and not containing any A's or X's.

AE - Alphanumeric Edited - An item whose PICTURE contains at least one of the insertion character B, and at least one X.

AN - Alphanumeric - The item has CLASS ALPHANUMERIC, but is neither Numeric Edited nor Alphanumeric Edited.

AB - Alphabetic - The item has CLASS ALPHABETIC. Any item whose PICTURE consists entirely of A's and B's is considered ALPHABETIC. (There must be at least one A.)

Numeric literals belong to the category N, and non-numeric literals belong to the category AN.

b) The following rules apply to an elementary MOVE between the categories defined above:

- (1) It is illegal to MOVE an NE, AE, or AB item to an N or NE item.
- (2) It is illegal to MOVE an N or an NE item to an AB item.
- (3) It is illegal to MOVE an N item whose implicit decimal point is not immediately to the right of the least significant digit to an AN or AE item.
- (4) A literal cannot be moved to a DISPLAY-2 item. A DISPLAY-2 item can be moved only to a DISPLAY or DISPLAY-2 item. When a DISPLAY-2 item is a receiving item, the sending item must have DISPLAY or DISPLAY-2 USAGE.
- (5) It is illegal to move a literal to a smaller field.
- (6) All other elementary moves are legal and are performed according to the rules given in Note 3.

3. The following rules apply to legal elementary moves:

- a) When an AE, AN, or AB item is a receiving item, justification and any necessary space-filling takes place as defined under the JUSTIFIED clause. If the SIZE of the sending item is greater than the SIZE of the receiving item, the excess characters are truncated after the receiving item is filled.
- b) When an N or NE item is a receiving item, alignment by decimal point and any necessary zero-filling takes place as defined under the JUSTIFIED clause, except where zeroes are placed because of editing requirements. If the receiving item has no operational sign, the absolute value of the sending item is used. If the sending item has more digits to the left and/or right of the decimal point than the receiving item can contain, the excess digits are truncated. If the sending item contains any non-numeric characters, the results at object time may be unpredictable.
- c) Whenever any excess characters are to be truncated, a warning will be given during compilation.
- d) Any necessary conversion of data from one form of internal representation to another (binary to decimal, numeric mode to alphanumeric mode, etc.) will take place during the MOVE, along with any specified editing in the receiving item such as suppress zeros with blanks, insert a dollar sign, commas, a decimal point, etc.

4. Any MOVE that is not an elementary MOVE is treated exactly as if it were an AN to AN elementary MOVE, except that there will be no conversion of data from one form of internal representation to another.
5. If the CORRESPONDING option is used, selected items within data-name-1 are moved, with any required editing, to selected areas within data-name-2. Items are selected by matching the data-names of items defined within data-name-1 with like data-names of areas defined within data-name-2, according to the following rules:
 - a) At least one of the items must be an elementary item.
 - b) The respective data-names are the same including all qualifications up to but not including data-name-1 and data-name-2.

Each CORRESPONDING source item is moved in conformity with the description of the receiving area. The results are the same as if the user had referenced each pair of CORRESPONDING data-names in separate MOVE statements.

6. In determining which data-names are CORRESPONDING to each other, any data-names which are subordinate to data-name-1 or data-name-2 and which have REDEFINES clauses are ignored, as well as any data-names which are subordinate to the subordinate data-names with REDEFINES clauses.

Note that this restriction does not preclude data-name-1 or data-name-2 themselves from having REDEFINES clauses or from being subordinate to data-names with REDEFINES clauses.

In the execution of "MOVE CORRESPONDING ABLE TO BAKER," where the respective data descriptions are as follows:

03 ABLE	03 BAKER
04 P	04 P
04 Q	04 Q
04 R REDEFINES Q	04 R
05 S	05 S

P and Q will be moved, but R will not be moved (it is a REDEFINES), nor will S be moved (it is subordinate to a REDEFINES); similarly, if the REDEFINES had been in BAKER.

7. If the CORRESPONDING option is used, no items in the group referenced can contain an OCCURS clause.
8. When a MOVE CORRESPONDING references a group which contains RENAMES entries, these entries are not considered in the matching of items.
9. A MOVE CORRESPONDING must not reference items having level numbers 66, 77, or 88.
10. CORR is an abbreviation for CORRESPONDING.

MULTIPLY

FUNCTION: To multiply numeric data items and set the value of an item equal to the results.

MULTIPLY { data-name-1 } BY { data-name-2 [GIVING data-name-3] }
 { literal-1 } { literal-2 GIVING data-name-3 }
 [ROUNDED] [; ON SIZE ERROR any imperative statement]

Notes:

1. The data-names used must refer only to elementary items. If GIVING is used, data-name-3 is not used as an operand; hence, its format may contain editing symbols. In all other cases, the data-names used must refer to NUMERIC items only.
2. All rules specified under the ADD verb regarding the size of operands, presence of editing symbols in operands, the ON SIZE option, the ROUNDED option, the GIVING option, truncation, and the editing of results, apply to the MULTIPLY verb.
3. When GIVING is not used, the initial value of data-name-1 or literal-1 will be multiplied by the initial value of data-name-2. The value of the multiplier will be replaced by the product resulting from operation. Note that a literal must not be used as a multiplier.

NOTE

FUNCTION: To allow the programmer to write explanatory statements in the PROCEDURE DIVISION of his program.

NOTE...

Notes:

1. Following the word NOTE may appear any combination of the characters from the COBOL character set, provided the COBOL rules for punctuation and word and literal formation are observed.
2. If a NOTE sentence is the first sentence of a paragraph, the entire paragraph is considered to be commentary. Proper format rules for paragraph structure must be observed.
3. If NOTE is not the first word of a paragraph, the commentary ends with a period followed by a space.
4. NOTES are produced on the compiler listing, but do not affect the object program.
5. The word NOTE can only appear as the first word of a sentence.

OPEN

FUNCTION: To initiate the processing of files. Performs checking or writing of labels, and other input/output functions.

```

OPEN      { INPUT
              OUTPUT } file-name-1 [WITH NO REWIND] [, file-name-2
              [WITH NO REWIND]...] [ { INPUT
              OUTPUT } file-name-3 [WITH NO REWIND]
              [, file-name-4 [WITH NO REWIND ...] ]

```

Notes:

1. At least one file must be named when the OPEN verb is used.
2. The verb OPEN must be applied to all files, and must be executed prior to the first READ or WRITE for a file.
3. A second OPEN of a file cannot be executed prior to the execution of a CLOSE of the file.
4. The OPEN does not obtain or release the first data record. A READ or WRITE, respectively, must be executed to obtain or release the first data record. Data cannot be moved to the record area, nor can the record area be tested or referenced in any way until after the first READ on the specified file. Unless there is an APPLY PROCESS AREA for an optional file, no reference to the record area should be made if the file is not present.
5. The user's beginning label subroutine will be executed if one is specified by the USE verb.
6. If an input file has been designated as optional in the File-Control paragraph of the Environment Division, the object program will cause an interrogation for the presence or absence of this file to occur. If the file is not present, the file will not be opened, an indication of the absence of the file will occur, and an "end-of-file" signal will be sent to the input/output control system of the object program. Thus, when the first READ for this file is encountered, the "end-of-file" path for this statement will be taken.

PERFORM

FUNCTION: To depart from the normal sequence of procedures in order to execute a procedure a specified number of times, or until a condition is satisfied, and then return to the normal sequence.

Option 1:

PERFORM procedure-name-1 [THRU procedure-name-2]

Option 2:

PERFORM procedure-name-1 [THRU procedure-name-2] { data-name-1 } TIMES
integer-1

Option 3:

PERFORM procedure-name-1 [THRU procedure-name-2] UNTIL condition-1

Option 4:

PERFORM procedure-name-1 [THRU procedure-name-2] VARYING
data-name-2 FROM { data-name-3 } BY { data-name-4 }
literal-1 literal-2
UNTIL condition-1

Option 5:

PERFORM procedure-name-1 [THRU procedure-name-2] VARYING
subscript-name-1 FROM { data-name-5 } BY { data-name-6 }
integer-2 integer-3
UNTIL condition-2
[AFTER subscript-name-2 FROM { data-name-7 } BY { data-name-8 }
integer-4 integer-5
[AFTER subscript-name-3 FROM { data-name-9 } BY { data-name-10 }
integer-6 integer-7
UNTIL condition-3
UNTIL condition-4]]

Notes:

1. The range of a PERFORM starts with the first executable statement in procedure-name-1 and continues in logical sequence through the last executable statement of:

- a) procedure-name-2 if specified, or
- b) procedure-name-1 if procedure-name-2 is not specified.

If procedure-name-1 is a USE procedure, procedure-name-2 must not be specified.

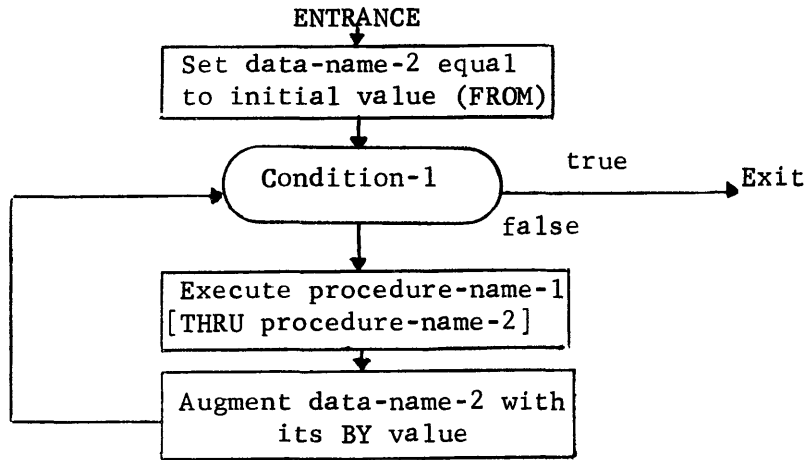
2. The first statement of procedure-name-1 is the point to which sequence control is sent by PERFORM. The return mechanism is automatically inserted as follows:

- a) If procedure-name-1 is a paragraph-name, and procedure-name-2 is not specified, -- after the last statement of the procedure-name-1 paragraph.
- b) If procedure-name-1 is a section-name, and procedure-name-2 is not specified, -- after the last statement of the last paragraph of the procedure-name-1 section.
- c) If procedure-name-2 is specified and is a paragraph-name, -- after the last statement of the procedure-name-2 paragraph.
- d) If procedure-name-2 is specified and is a section-name, -- after the last statement of the last paragraph of the procedure-name-2 section.

The "last sentence" PERFORMed in all of the above cases must allow control to pass to the return mechanism. There is no necessary relation between procedure-name-1 and procedure-name-2 except that a sequence beginning at procedure-name-1 must proceed through the last sentence of procedure-name-2. In particular, GO's and PERFORM's may occur between procedure-name-1 and the end of procedure-name-2, provided control eventually passes to the return mechanism. If it is desired to have two or more logical paths to the return mechanism, then procedure-name-2 must be a paragraph consisting of the verb EXIT to which these paths must lead.

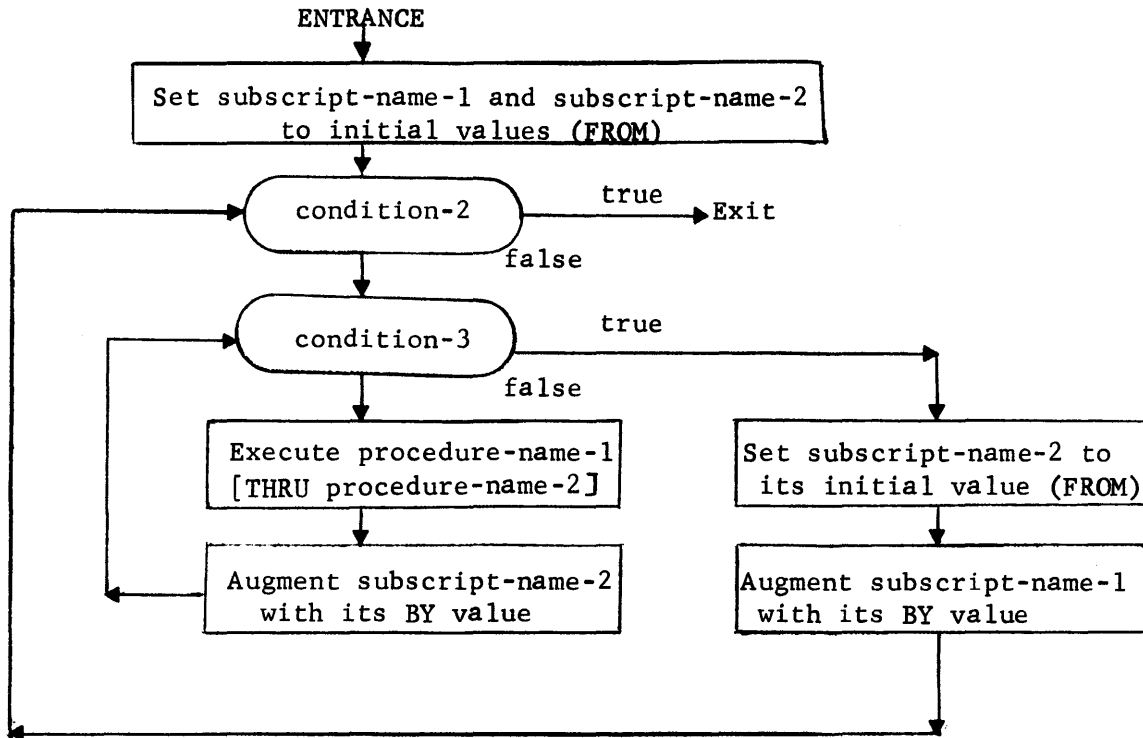
3. In all cases, after the completion of a PERFORM, a bypass is automatically created around the return mechanism which had been inserted after the "last statement." Therefore, when no related PERFORM is in progress, sequence control will pass through a "last statement," to the following statement as if no PERFORM had existed.

4. The **PERFORM** mechanism for options 1 through 4 operates as follows with note 3 above applying to all options:
- a) Option 1 is a simple **PERFORM**. A return to the statement following the **PERFORM** is inserted after the "last statement" as defined in note 2, and sequence control is sent to procedure-name-1.
 - b) Option 2 is the **TIMES** option. The specified number of times must be a positive integer, and may be zero. The **PERFORM** mechanism sets up a counter and tests it against the specified value before each jump to procedure-name-1. The return mechanism after the "last statement" steps the counter and then sends control to the test. The test cycles control to procedure-name-1 the specified number of times, and after the last time sends control to the statement following the **PERFORM**.
 - c) Option 3 is the **UNTIL** option. This option is the same as the **TIMES** option, except that an evaluation of a condition takes the place of counting and testing against a specified integer. The condition may be any simple or compound condition, that is, the condition may involve relations and tests. When the condition is satisfied, i.e., true, control is transferred to the next statement after the **PERFORM** statement. If the condition is true when the **PERFORM** is entered, no jump to procedure-name-1 takes place, and control is transferred to the next statement after the **PERFORM** statement.
 - d) Option 4 is the **VARYING** data-name option. The **VARYING** option is assumed to be arithmetic and all the arithmetic rules apply. This option is used when it is desired to increase or decrease the value of any item while the execution of a procedure or a series of procedures is being accomplished. Only one item can be varied per **PERFORM** statement using this option. The **PERFORM** mechanism sets the value of data-name-2 equal to its starting value (the **FROM**), then evaluates the condition (the **UNTIL**) for truth or falsity. If the condition is true at this point, then no execution of procedure-name-1 through procedure-name-2 takes place. Instead, control is transferred to the next statement after the **PERFORM** statement. If the condition is false, then procedure-name-1 through procedure-name-2 are executed once. The mechanism then augments the value of data-name-2 by the specified increment or decrement (the **BY**) and again evaluates the condition (the **UNTIL**) for truth or falsity. The cycle continues until the condition is determined to be true, at which point control is transferred to the next statement after the **PERFORM** statement. Literal-1 and literal-2 must be numeric literals, but need not necessarily be integral. A diagram for this mechanism follows.

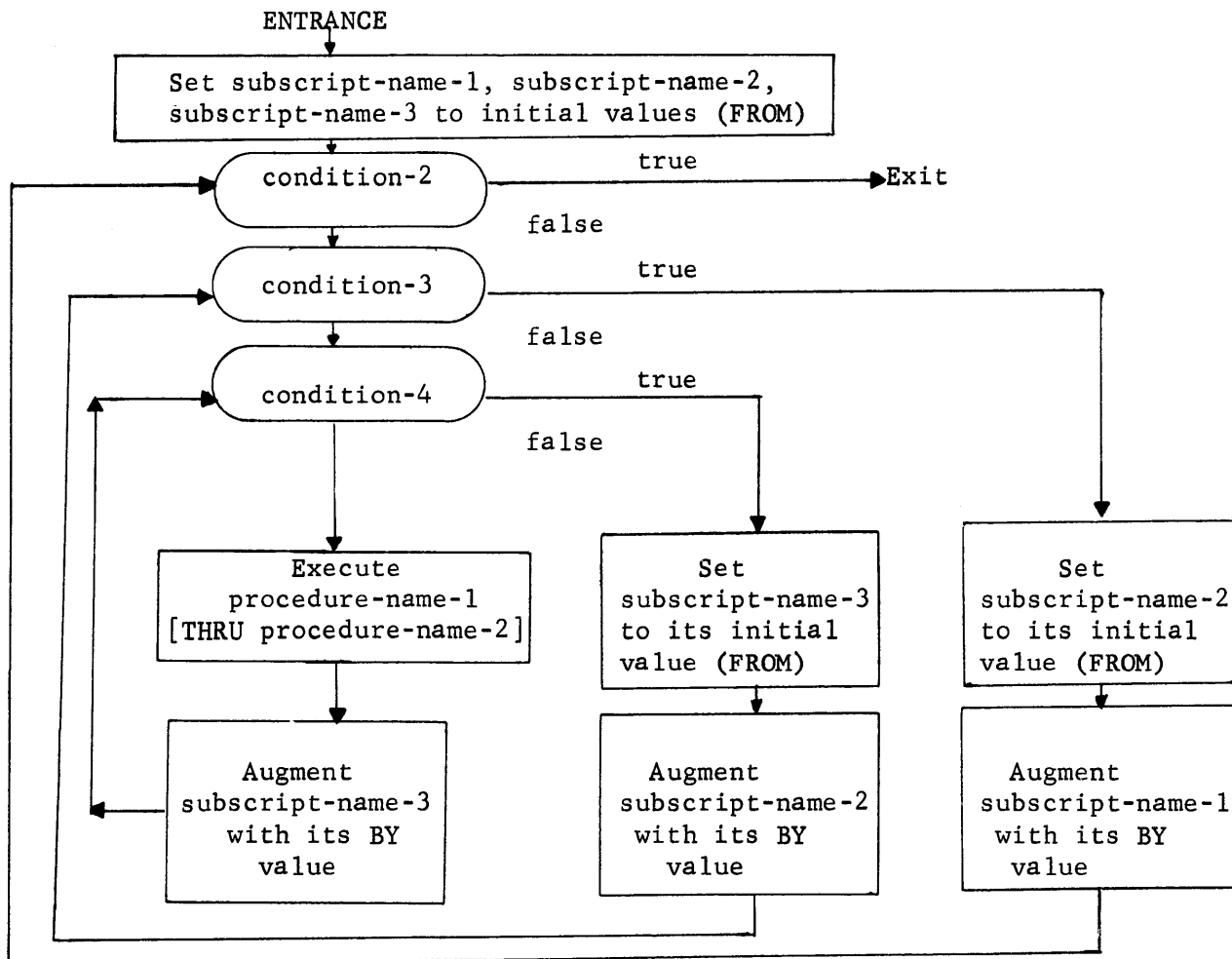


It should be noted that after the last execution of the PERFORM range, data-name-2 will be augmented one more time before control passes to the Exit of the PERFORM statement.

- e) Option 5 is the VARYING subscript-name option. This option is used when it is desired to augment the value of one or more subscripts in a nested fashion while the execution of a procedure or a series of procedures is being accomplished. A maximum of three subscripts can be varied per PERFORM statement using this option. When only one subscript is being varied, the mechanism is exactly the same as that of the VARYING data-name-2 option (Option 4). When two subscripts are varied, the value of subscript-name-2 goes through a complete cycle (FROM, BY, UNTIL) each time that subscript-name-1 is augmented with its BY value. The PERFORM is completed as soon as condition-2 is found to be true. When three subscripts are varied the value of subscript-name-3 goes through a complete cycle (FROM, BY, UNTIL) each time that subscript-name-2 is augmented with its BY value. Further, subscript-name-2 goes through a complete cycle (FROM, BY, UNTIL) each time that subscript-name-1 is augmented by its BY value. The PERFORM is completed as soon as condition-2 is found to be true. Regardless of the number of subscripts being varied, as soon as condition-2 is found to be true, control is transferred to the next statement after the PERFORM statement. The FROM value must be a positive, non-zero integer. The BY value must be a non-zero integer. Subscript-name-1, subscript-name-2, and subscript-name-3 must never reference the same item (i.e., they must not be alternative names for the same data item). Diagrams for this mechanism follows.

TWO SUBSCRIPTS

THREE SUBSCRIPTS

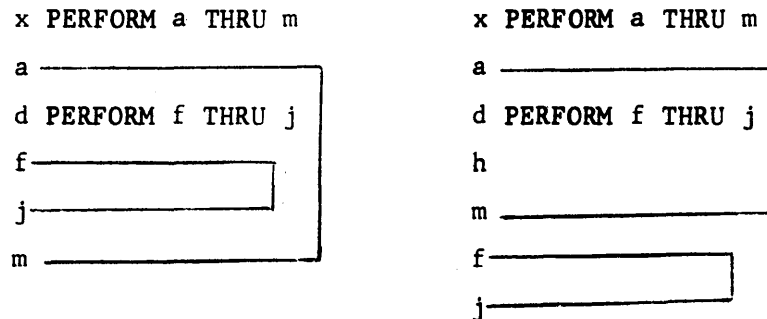


It should be noted that after the last execution of the PERFORM range subscript-name-2 and subscript-name-3 will each be set to their respective initial value (FROM) while subscript-name-1 will be augmented one more time before control passes to the Exit of the PERFORM statement.

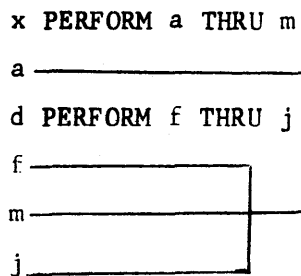
5. In general, procedure-name-1 should not be the next statement after the PERFORM. If it is, the result will be that the loop will be executed one more time than was probably intended, because after the PERFORM is satisfied control would go to procedure-name-1 in the normal continuation of the sequence.

6. If a sequence of statements referenced by a PERFORM includes another PERFORM statement, the sequence associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referenced by the first PERFORM.

For example, the following illustrations are correct.

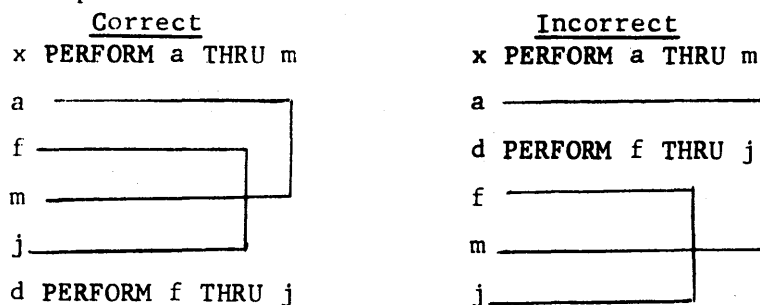


The following illustration is incorrect:



The sequence of procedures associated with a PERFORM statement may overlap or intersect the sequence associated with another PERFORM provided that neither sequence includes the PERFORM statement associated with the other sequence.

For example:



READ

FUNCTION: To make available the next logical record from an input file and to allow performance of a specified imperative statement when end-of-file is detected.

READ file-name RECORD [INTO data-name] [; AT END any imperative statement]

Notes:

1. An OPEN statement for the file must be executed prior to the execution of the first READ for that file.
2. When a file consists of more than one type of logical record, these records automatically share the same storage area. This is equivalent to saying that there exists an implicit redefinition of the area, and only the information which is present in the current record is accessible.
3. No reference can be made by any verb in the PROCEDURE DIVISION to information which is not actually present in the current record. If such a reference is made the results in the object program are unpredictable.
4. The INTO data-name option may only be used when the input file contains just one type of record. The data-name must be the name of a working storage or output record area. If the format of the data-name differs from that of the input record, moving will be performed according to the rules specified for the MOVE verb without the CORRESPONDING option. When the INTO data-name option is used, the "file-name RECORD" is still available in the input record area.
5. Every READ statement must have an END of file option, either implicitly or explicitly. On the next execution of a READ statement for the file after the last record in the file has been made available, "any imperative statement" is executed. If the user does not write END, the compiler will examine all other READ statements for the same file. If the word END appears once and only once for a given file, the compiler will append this and its associated imperative-statement-1 to each READ statement for that file which has no explicit END of file option. If more than one, but not all READs for the same file contain the word END in their formats, the compiler will indicate an error during compilation.

6. If an OPTIONAL file is not present, the imperative statement will be executed on the first READ. The standard end-of-file procedures will not be performed. (See the OPEN and USE verbs, and the FILE-CONTROL paragraph in the ENVIRONMENT DIVISION.)
7. After execution of the imperative statement, an attempt to perform a READ without the execution of a CLOSE and a subsequent OPEN for that file would normally constitute an error in the object program. However, in special cases it may be desirable to READ a second reel of an unlabeled multiple reel file. (See Processing Nonlabeled Multiple Reel Files.)
8. If an end of reel is recognized during execution of a READ statement, the following operations are carried out:
 - a) The standard ending reel label procedure and the user's ending reel label procedure (if specified by the USE verb). The order of execution of these two procedures is specified by the USE verb.
 - b) A tape swap. (This includes rewinding the exhausted reel and placing it in standby status.)
 - c) The standard beginning reel label procedure and the user's beginning reel label procedure (if specified by the USE verb). The order of execution of these two procedures is specified by the USE verb.
 - d) The first data record on the new reel is made available.

RELEASE

FUNCTION: To transfer records to the initial phase of a SORT operation.

RELEASE record name [FROM data-name]

Notes:

1. RELEASE can only be used within an INPUT PROCEDURE associated with a SORT statement for a file whose DATA RECORDS clause contains record-name. Any other use of a RELEASE statement will lead to unpredictable results.
2. Record-name must be named in the DATA RECORDS clause of its associated sort-file.
3. Data-name must be the name of a working storage or an input record area. If the format of data-name differs from that of the record-name, moving will take place according to the rules specified for the MOVE verb without the CORRESPONDING option. The information in the record area is no longer available, but the information in data-name area is available. It is illegal to use the same name for both data-name and record-name.
4. After the RELEASE is executed, record-name is no longer available.
5. The execution of a RELEASE statement causes record-name (after data-name has been MOVED to it in the FROM option) to be transferred to the initial phase of a SORT. When control passes from the INPUT PROCEDURE the file consists of all those records which were placed in it by the execution of RELEASE statements. No OPEN, CLOSE, READ, WRITE, or USE statements may be given for the sort-file.

RETURN

FUNCTION: To obtain sorted records from the final phase of a SORT operation.

RETURN file-name RECORD [INTO data-name] [; AT END any imperative statement]

Notes:

1. File-name must be a sort-file with a sort-file description in the DATA DIVISION.
2. RETURN can only be used within an OUTPUT PROCEDURE associated with a SORT statement for file-name. Any other use of a RETURN statement will lead to unpredictable results at object time.
3. The execution of the RETURN statement causes the next record in sorted order (according to the keys listed in the SORT statement) to be made available for processing in the records area associated with the sort file. No OPEN, READ, WRITE, CLOSE or USE statements may be given for the sort-file.
4. The INTO data-name option may only be used when the input file contains just one type of record. The data-name must be the name of a working storage or output record area. If the format of the data-name differs from that of the input record, moving will be performed according to the rules specified for the MOVE verb without the CORRESPONDING option. When the INTO data-name option is used, the "file-name RECORD" is still available in the input record area.
5. Every RETURN statement must have an END of file option, either explicitly or implicitly. If the user does not write END, the compiler will examine all other RETURN statements within this OUTPUT PROCEDURE. If the word END appears once and only once, the compiler will append this and its associated imperative-statement-1 to each RETURN statement. If more than one, but not all RETURN statements within this OUTPUT PROCEDURE contain the word END, an error will be indicated during compilation.
6. After execution of "any imperative statement" no RETURN statement may be executed within the current OUTPUT PROCEDURE. The results of such an error will be unpredictable.

SORT

FUNCTION: To create a sort-file by executing input procedures or by transferring records from another file, to sort the records in the sort-file on a set of specified keys, and in the final phase of the sort operation, to make available each record from the sort-file, in sorted order, to some output procedures or to an output file.

```

SORT file-name-1 ON { DESCENDING } KEY data-name-1 [, data-name-2...]
                   { ASCENDING }
[; ON { DESCENDING } KEY ...]
      { ASCENDING }
{ INPUT PROCEDURE IS section-name-1 [THRU section-name-2] }
{ USING file-name-2 }
{ OUTPUT PROCEDURE IS section-name-3 [THRU section-name-4] }
{ GIVING file-name-3 }

```

Notes:

1. File-name-1 must have a Sort-file Description in the DATA DIVISION.
2. The data-name KEYS are listed from left to right in the SORT statement in order of significance without regard to how they are divided into KEY clauses. In the format data-name-1 is the major key, data-name-2 is the next most significant key. Key data-names must not be subscripted.
 - a) When an ASCENDING clause is used, the sorted sequence will be from lowest value of key to highest value according to the rules given under Simple Conditions.
 - b) When a DESCENDING clause is used, the sorted sequence will be from highest value of key to lowest value according to the rules given under Simple Conditions.
3. Every record which is listed in the DATA RECORDS clause of the Sort-file must contain within its Record Description the KEY items data-name-1, data-name-2, etc., and each of the KEY items must have the same relative position and the same description in every one of the records. When more than one record is present, the first record determines the "dominant record size" parameter for the sort. (See GE-625/635 SORT/MERGE, CPB-1005.)
4. If an INPUT PROCEDURE is specified, control is passed to the INPUT PROCEDURE before file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last Section in the INPUT PROCEDURE and when control passes from the last statement in the INPUT PROCEDURE, the records which have been RELEASED to file-name-1 will be sorted.

5. The INPUT PROCEDURE must consist of one or more Sections which must be written consecutively, and do not form a part of any OUTPUT PROCEDURE. The INPUT PROCEDURE must include at least one RELEASE statement in order to transfer records to the sort-file. Control must not be passed to the INPUT PROCEDURE except when a related SORT statement is being executed. The INPUT PROCEDURE can include any procedures needed to select, create or modify records. There are three restrictions on the procedural statements within the INPUT PROCEDURE:
 - a) The INPUT PROCEDURE must not contain any SORT statements.
 - b) The INPUT PROCEDURE must not contain any transfers of control to points outside the INPUT PROCEDURE: i.e., ALTER, GO and PERFORM statements in the INPUT PROCEDURE are not permitted to refer to procedure-names outside the INPUT PROCEDURE.
 - c) The remainder of the PROCEDURE DIVISION must not contain any transfers of control to points inside the INPUT PROCEDURE i.e., ALTER, GO and PERFORM statements in the remainder of the PROCEDURE DIVISION are not permitted to refer to procedure-names within the INPUT PROCEDURE.
6. If the USING file-name-2 option is specified, this implies that all the records in file-name-2 are transferred automatically to file-name-1. At the time of execution of the SORT statement, file-name-2 must not be OPEN. The SORT statement will automatically perform the necessary OPEN, READ, and CLOSE functions for file-name-2. File-name-2 must have a file-description, not a sort-file description in the DATA DIVISION. The DATA RECORDS of file-name-2 and their descriptions must be identical to those of file-name-1. USE procedures cannot be applied to file-name-2 during the Sort.
7. If an OUTPUT PROCEDURE is specified, control passes to it after file-name-1 has been sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last Section in the OUTPUT PROCEDURE and when control passes from the last statement in the OUTPUT PROCEDURE, the return mechanism will provide for termination of the sort and then will send control to the next statement after the SORT statement. Before entering it can select the next record in sorted order when requested. The RETURN statements in the OUTPUT PROCEDURE are the requests for the next record.

8. The OUTPUT PROCEDURE must consist of one or more Sections which must be written consecutively, and do not form a part of any INPUT PROCEDURE. The OUTPUT PROCEDURE must include at least one RETURN statement in order to make sorted records available for processing. Control must not be passed to the OUTPUT PROCEDURE except when a related SORT statement is being executed. The OUTPUT PROCEDURE can consist of any procedures needed to select, modify or copy the records which are being RETURNed one at a time in sorted order, from the sort-file. There are three restrictions on the procedural statements within the OUTPUT PROCEDURE:
 - a) The OUTPUT PROCEDURE must not contain any SORT statements.
 - b) The OUTPUT PROCEDURE must not contain any transfers of control to points outside the OUTPUT PROCEDURE: i.e., ALTER, GO and PERFORM statements in the OUTPUT PROCEDURE are not permitted to refer to procedure-names outside the OUTPUT PROCEDURE.
 - c) The remainder of the PROCEDURE DIVISION must not contain any transfers of control to points inside the OUTPUT PROCEDURE: i.e., ALTER, GO and PERFORM statements in the remainder of the PROCEDURE DIVISION are not permitted to refer to procedure-names within the OUTPUT PROCEDURE.

9. If the GIVING option has been used, this means that all the sorted records in file-name-1 are automatically transferred to file-name-3 as the implied output procedure for this SORT statement. At the time of execution of the SORT statement file-name-3 must not be OPEN. File-name-3 is automatically OPENed before transferring the records and a CLOSE file-name-3 is executed automatically after the last record in the sort-file is RETURNed. File-name-3 must have a File Description, not a Sort-file Description, in the DATA DIVISION. The DATA RECORDS of file-name-3 and their descriptions must be identical to those of file-name-1. USE procedures cannot be applied to file-name-3 during a sort.

STOP

FUNCTION: To halt the object program either permanently or temporarily.

STOP { literal
 RUN }

Notes:

1. If the word RUN is used, then the standard ending procedure is instituted.
2. The literal will be presented on a console typewriter when the STOP literal option is used. The literal must satisfy the rules for operands stated in the notes under the DISPLAY verb. Use of this option is strongly discouraged, except in extraordinary circumstances. Under no circumstances should it be used to terminate execution of a program.

SUBTRACT

FUNCTION: To subtract one, or the sum of two or more, numeric data items from an item, and set the value of an item equal to the results.

Option 1:

SUBTRACT { literal-1
data-name-1 } [, { literal-2
data-name-2 } . . .] FROM
{ literal-n GIVING data-name-m
data-name-n [GIVING data-name-m] } [ROUNDED]
[; ON SIZE ERROR any imperative statement]

Option 2:

SUBTRACT { CORRESPONDING
CORR } data-name-1 FROM data-name-2 [ROUNDED]

Notes:

1. In option 1, the data-names used must refer only to elementary items. If GIVING is used, data-name-n is not used as an operand; hence, its format may contain editing symbols. In all other cases, the data-names used must refer to NUMERIC items only.
2. All rules specified under the ADD verb with respect to the size of operands, presence of editing symbols in operands, the ON SIZE ERROR option, the ROUNDED option, the GIVING option, truncation, and the editing of results and the CORRESPONDING option apply to the SUBTRACT verb.
3. When the GIVING option is not used, a literal may not be specified as the minuend.
4. When dealing with multiple subtrahends, the effect of the subtraction will be as if the subtrahends were first summed, and the sum was then subtracted from the minuend.
5. CORR is an abbreviation for CORRESPONDING.

TERMINATE

FUNCTION: To TERMINATE the processing of a report.

TERMINATE {data-name-1 [, data-name-2 ...] }
ALL

Notes:

1. TERMINATE produces all the CONTROL FOOTINGS associated with this report as if a CONTROL break had just occurred at the highest level, i.e., FINAL CONTROL break, and completes the Report Writer functions for the named reports.
2. Appropriate PAGE and OVERFLOW HEADING and/or FOOTING report groups are prepared in their respective order for the report description.
3. Each data-name given in a TERMINATE must be defined by an RD entry in the Data Division.
4. A second TERMINATE for a particular report cannot be executed.
5. If ALL is specified, all reports defined in the Report Section of the Data Division which were INITIATED are TERMINATED.
6. TERMINATE does not close the file with which the report is associated. A CLOSE statement for the file must be executed after the TERMINATE statement has been executed.

USE

FUNCTION: To specify procedures for any computer I/O error and label handling which are in addition to the standard procedures supplied by the input/output system; and to specify PROCEDURE DIVISION statements which are executed just before a named report group in the REPORT SECTION, DATA DIVISION, is presented.

Option 1:

USE AFTER STANDARD ERROR PROCEDURE ON { file-name
INPUT } .

Option 2:

USE { BEFORE
AFTER } STANDARD { BEGINNING
ENDING } LABEL PROCEDURE ON { file-name
INPUT
OUTPUT } .

Option 3:

USE BEFORE REPORTING data-name-1.

Notes:

1. A USE sentence, when present, must immediately follow a Section header in the DECLARATIVES. The remainder of the Section must consist of one or more procedural paragraphs which define the procedures to be used.
2. The designated procedures will be executed by the input/output system at the appropriate time, that is:
 - a) After completing the standard I/O error routine. (This applies only to Option 1.)
 - b) Before or after a beginning or ending input label check procedure is accomplished. (Applies only to Option 2.)
 - c) Before a beginning or ending output label is created. (Applies only to Option 2.)
 - d) After a beginning or ending output label is created, but not written on tape. (Applies only to Option 2.)
3. When Option 2 is used:
 - a) If the file-name option is used, the File Description entry for file-name must not specify LABEL RECORDS ARE OMITTED.

- b) If the INPUT (or OUTPUT) option is used, the procedures will not be executed for any INPUT (or OUTPUT) file whose File Description specifies LABEL RECORDS ARE OMITTED.
 - c) If BEGINNING or ENDING is not included, the designated procedures will be executed for both beginning and ending labels.
 - d) The designated procedures will be executed for both REEL and FILE labels.
- 4. In Option 3, the designated procedures are executed by the object program just before the named report group is produced, regardless of page, overflow, and/or control break associations with report groups. A report group cannot be referenced by more than one USE statement. Data-name-1 must be a non-detail item.
 - 5. No Report Writer verb (GENERATE, INITIATE, or TERMINATE), input/output verb (OPEN, READ, WRITE, or CLOSE), or SORT verb (SORT, RETURN, or RELEASE) may be written in Use procedures. A USE section may contain PERFORM statements referencing other USE procedures, but it may not contain any other references to procedures outside itself.
 - 6. Option 2, Beginning Label Procedures must not contain any DISPLAY, DISPLAY SYSOUT, or ACCEPT GEIN statements.

definitely fail to meet printer requirements, and the printout will be unsatisfactory. In the ADVANCING option, the following rules apply:

- a. When data-name-2 is specified, it must be a non-negative COMPUTATIONAL-1 item whose size does not exceed 8 digits. When the WRITE statement is executed, the value of data-name-2 will determine the number of lines the listing is advanced.
- b. When integer is specified, it must be non-negative. Its value will determine the number of lines the listing is advanced.
- c. TOP causes the listing to be advanced to top of page.
- d. The ADVANCING option may be used only when FOR LISTING has been specified in the SELECT sentence in the ENVIRONMENT DIVISION. It cannot be used for a file described with OCCURS...DEPENDING. System Standard Format should generally be specified for a file to which WRITE...ADVANCING is applied.

VIII. ENVIRONMENT DIVISION

GENERAL DESCRIPTION

The ENVIRONMENT DIVISION is used to specify the aspects of the program which relate to the physical characteristics of the computer. It consists of two sections - CONFIGURATION and INPUT-OUTPUT.

The CONFIGURATION SECTION provides the overall description of the computer. It consists of three paragraphs. The SOURCE-COMPUTER paragraph describes the computer on which the COBOL compiler is to be run. The OBJECT-COMPUTER paragraph describes the computer on which the object-program is to be run. The SPECIAL-NAMES paragraph associates mnemonic-names with report codes and special system features.

The INPUT-OUTPUT SECTION provides information needed for efficient object-program transmission and handling of data between the external media and the computer memory. This section consists of two paragraphs. The FILE-CONTROL paragraph associates each file with an external medium. The I-O CONTROL paragraph specifies rerun options, special input/output techniques, and files which appear on multiple file tapes.

STRUCTURE

The following illustration shows the fixed section and paragraph names of the ENVIRONMENT DIVISION in the order in which they must appear in the source program. A section or paragraph may optionally be omitted if it is not needed.

```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  GE-635...
.
.
.
OBJECT-COMPUTER.  GE-635 ...
.
.
SPECIAL-NAMES.  ...
INPUT-OUTPUT SECTION.
FILE-CONTROL.  SELECT ...
.
.
I-O-CONTROL.  APPLY ...
.
.

```

CONFIGURATION SECTION.

SOURCE-COMPUTER.

FUNCTION: To describe the computer upon which the program is to be compiled.

Option 1:

SOURCE-COMPUTER. COPY library-call.

Option 2:

SOURCE-COMPUTER. { GE-625 }
 { GE-635 }

[, MEMORY SIZE integer-1 WORDS]

[, [integer-2] hardware-name-1 [, [integer-3] hardware-name-2..]].

Notes:

1. The configuration may include more equipment than is actually needed by the compiler.
2. Integer-2 (if present) specifies the number of units of hardware-name-1; similarly for integer-3, etc.
3. Hardware-names must be taken from the following list of reserved words:

MAGNETIC TAPE UNIT(S)
DISC STORAGE UNIT(S)
DRUM STORAGE UNIT(S)
HIGH SPEED PRINTER(S)
CARD PUNCH(ES)
CARD READER(S)
INPUT-OUTPUT TYPEWRITER(S)

OBJECT-COMPUTER.

FUNCTION: To describe the computer upon which the object-program is to be executed.

Option 1:

OBJECT-COMPUTER. COPY library-call.

Option 2:

OBJECT-COMPUTER. { GE-625
GE-635 }

[, MEMORY SIZE integer-1 WORDS]

[, [integer-2] hardware-name-1 [, [integer-3] hardware-name-2. . .]].

Notes:

1. The configuration may include more equipment than is actually needed by the object-program.
2. Integer-2, if present, specifies the number of units of hardware-name-1; similarly for integer-3, etc.
3. Hardware-names must be taken from the following list of reserved words:

MAGNETIC TAPE UNIT(S)
DISC STORAGE UNIT(S)
DRUM STORAGE UNIT(S)
HIGH SPEED PRINTER(S)
CARD PUNCH(ES)
CARD READER(S)
INPUT-OUTPUT TYPEWRITER(S)

SPECIAL-NAMES.

FUNCTION: To associate mnemonic-names (external-names) with report-codes or special system features.

Option 1:

SPECIAL-NAMES. literal-1 IS mnemonic-name-1

[, literal-2 IS mnemonic-name-2. . .].

Option 9:SPECIAL-NAMES.

ELECT SORT OPTIONS field-1, field-2, field-3, field-4, field-5,
field-6, field-7, field-8, field-9, field-10, field-11, field-12,
FOR file-name-1 [, file-name-2].

Notes:

1. This paragraph may be omitted when its provisions are not used in the source-program.
2. Option 1 is used to define report-codes, for use via the CODE clause in RD entries in the Report Section of the Data Division. Each literal must be a single-character non-numeric literal whose value is a letter (A,--Z) or a digit (0,--9).
3. Option 2 is used to associate mnemonic-names with GETIME, GELAPS, or the input-output TYPEWRITER or CONSOLE. GETIME is a GECOS feature which supplies current date and time upon request. (See the GECOS manual.) GELAPS is a GECOS feature which supplies elapsed time charged thus far to the requesting program. The special-names in option 2 can only be GEIN, SYSOUT, GETIME, GELAPS TYPEWRITER, or CONSOLE. A mnemonic-name associated with GEIN, GETIME, or GELAPS can be referenced only in ACCEPT statements. A mnemonic-name associated with the TYPEWRITER or CONSOLE can only be referenced by ACCEPT or DISPLAY statements. A mnemonic-name associated with SYSOUT can be referenced only in DISPLAY statements. For a detailed explanation of the use of these features, see the notes under ACCEPT and DISPLAY in Chapter VII.
4. Option 3 is used to associate mnemonic-names with switches and/or to associate condition-names with specific switch settings. Each integer must be in the range 0, 1, -35. An ON condition-name corresponds to a switch value of 1, while an OFF condition-name corresponds to a switch value of 0. Switches 0-5 and 18-35 may be used for communication between activities. Switches 12-17 are reserved for the operating system. Switches 6-11 are reset at the start of each activity, on the basis of the ON option of the \$EXECUTE card (see the GECOS manual). Each of switches 6-11 is set "off" at that time unless the \$EXECUTE card causes it to be set "on," according to the following table:

<u>\$EXECUTE</u> <u>card parameter</u>	<u>results in "on"</u> <u>setting for switch</u>
ON1	6
ON2	7
ON3	8
ON4	9
ON5	10
ON6	11

The mnemonic-names associated with switches may be referenced only in ACCEPT and DISPLAY statements. Condition-names may be referenced only in conditions appearing in IF or PERFORM statements. In Option 3, at least one of the optional phrases must be used with each switch specified.

The switches are software feature provided by GECOS, not actual hardware switches.

5. Option 4 is used to cause WORKING-STORAGE or CONSTANT SECTION storage areas to be allocated in the form of labeled storage blocks, for purposes of source-program segmentation. (See Chapter X.) The contents of a labeled storage block are accessible to each source-program segment in which the block is described.

Integer-1 must be a 1- or 2-digit integer, which serves as a unique label for the storage block.

Data-name-1 must be a level 01 or 77 item in WORKING-STORAGE or CONSTANT SECTION. Data-name-2 must be a level 01 or 77 item in the same section. Data-name-2 must appear later than data-name-1 in the source-program.

BLOCK statements must appear in the same order as the items they refer to.

6. Each mnemonic-name must be a unique word originated by the user, satisfying the rules for data-name formation but not given a description in the Data Division.
7. In Option 5, the respective clauses have the following significance:
 - a. COLLATE COMMERCIAL causes a special provision to be made for comparisons in the object-program. When Alphanumeric items are compared, the result is based on the commercial collating sequence (p. VII-8) instead of the standard collating sequence of the machine. This provision applies to SORT keys as well as to conditional statements.
 - b. DECIMAL-POINT IS COMMA causes the function of the comma and the period to be exchanged in PICTURE and in numeric literals. Rules for PICTURE formation and numeric literal formation change accordingly. (This COBOL feature accommodates conventional notation for various European currencies.)
 - c. The CURRENCY SIGN clause permits an arbitrary single character to replace the dollar sign in PICTURE and in object-program editing. The literal must be a single-character non-numeric literal, and must not be any of the following PICTURE characters:

0-9

A, B, C, D, J, K, L, P, R, S, V, X, or Z

*, +, -, ., or comma (,)

8. Options 6 and 7 relate only to special statements in the source program identified by one of the digits 0 through 9 in Column 7. The use or omission of either of these options has no effect whatever on standard source statements with a blank, hyphen or asterisk in Column 7. However, do not attempt continuation to the next line (hyphen in Column 7) of an optional COMPILE statement. A detailed description of this special compiler feature is provided in Appendix I.
9. Option 6 is used to request compilation of all source statements identified by any one of the digits 0 through 9 in Column 7.
10. In Option 7 one of two selections may be made:
 - a. If only integer-1 is specified, then those source statements with the "integer-1" identifier in Column 7 will be compiled.
 - b. If integer-1 THRU integer-2 is specified, all source statements with an identifier in Column 7 within the range of "integer-1 THRU integer-2" inclusive will be compiled.
11. If neither Option 6 nor Option 7 is specified, then all source statements identified by any one of the digits 0 through 9 in Column 7 will be unconditionally bypassed during compilation.
12. Option 8 is used to optimize the compilation process by eliminating the output of an assembly listing and object program, in the event of source program errors being encountered. A detailed description of this special feature is provided in Appendix K.
13. When Option 8 is not selected, an assembly listing and object program are produced unconditionally.
14. Option 9 is used to modify the standard functions of the Sort program for the specified COBOL files which are described using an SD file description. Twelve options are provided corresponding to the ELECT Macro as described for the free-standing Sort system. The programmer must preserve the order of the fields according to their numbers as presented below. When the ELECT option is used, all fields must be specified according to field description or as a null field. If file-name-2 is specified, it will receive the same options as file-name-1. There must be no more than one ELECT for any given sort-file.

a. Field-1: Output Order Control

This parameter may be used to establish a tie-breaking convention for Sort key comparisons. Acceptable values are:

- | | | |
|---|---|---|
| 0 | - | The output order of equal key data records will be indeterminate. |
|---|---|---|

- 1 - The original input order of the data will be used as the basis for breaking ties between equal keys. Selection will be made on the basis of original input sequence.

b. Field-2: Error Journal Control

This parameter overrides the standard journalization option in case of input or output errors. Acceptable values are:

- 0,N or null - Retain full journalization capabilities. (See the Sort Manual for description.)
- T - Journalize error and then terminate processing through the GEBORT routine of GECOS.
- D - Do not journalize errors but continue processing. Although there is no journalization, an error occurrence message is placed on the execution report (SYSOUT).
- A - Do not journalize errors. After placing an error occurrence message on the execution report, terminate processing through the GEBORT routine of GECOS.
- 1 - Journalize errors. If the error occurred while reading the original input, continue processing. If the error occurred while reading a collation file, terminate processing through the GEBORT routine of GECOS.
- 2 - Journalize errors. If the error occurred while reading the original input, continue processing. If the error was block serial number error while reading a collation file, terminate processing through the GEBORT routine of GECOS.
- 3 - Journalize errors. If the error was a block serial number error while reading the original input, terminate processing through the GEBORT routine of GECOS. If any error occurred while reading a collation file, terminate processing through the GEBORT routine of GECOS.

- 4 - Journalize errors. If the error was a block serial number error while reading either the original input or a collation file, terminate processing through the GEBORT routine of GECOS.

The latter four options allow the user to discriminate between original input errors and collation file errors. They also allow different recovery options for general read errors and block serial number errors. This differentiation may be required because the journalization of a block serial error cannot contain the text of the missing block. If such an error occurs on a collation file, it is impossible to determine what records were dropped.

c. Field-3: Checkpoint Control

This parameter controls the taking of checkpoint records during a Sort or Merge process. Acceptable values are:

- 0 - Do not take checkpoint records.
or
null
- 1 - Take checkpoint records. During the Sort process, checkpoints are written on whichever file is the current output file unless a RERUN ON file-name clause has been specified for the sort-file.

d. Field-4: Logical Record Memory Assignment

This parameter forces the first word of a data record into an even or odd memory location during a sorting process. This option is normally used to optimize comparison coding for double precision keys. Acceptable parameter values are:

- 0 - The placement of logical records in memory is indeterminate.
or
null
- 1 - The first word of every record being sorted is placed in an odd memory location.
- 2 - The first word of every record being sorted will be placed in an even memory location.

e. Field-5: Borrow Tapes Control

The use of this parameter allows the dynamic allocation of free tape handlers to a Sort process at execution time. Such tapes are used for collation files in addition to the tapes allocated by \$ TAPE or \$ NTAPE cards. Acceptable parameter values are:

- 0 - The Sort process does not borrow tape for collation files.
- or null
- n - The n represents a numeric value from 1 to 13. The Sort process borrows up to n tapes for collation files during execution. Borrowed tapes are released following the final collation pass.

f. Field-6: Input Device Positioning Control

This field allows the user to specify handling of the input file while opening and closing the file. The user is responsible for selecting appropriate disposition codes on file control cards. Acceptable values are as follows:

	OPEN	CLOSE	
	Rewind	Rewind	Lock
0 or Null	Yes	Yes	No
1	Yes	Yes	Yes
2	Yes	No	No
3	Yes	Yes	No
5	No	Yes	Yes
6	No	No	No
7	No	Yes	No

g. Field-7: Output File Collation Control

This parameter controls the use of the output file as a collation file during a sorting process. Normally, the output file is so used. However, if the output file is not a tape device, this parameter must be used to inhibit its use as a collation file. Acceptable values are:

0 - Use the output file as a collation working file.
or
null

1 - Do not use the output file as a collation working file.

h. Field-8: Output Device Positioning Control

This field allows the user to specify special handling of the output file while opening and closing the file. The user is responsible for selecting appropriate disposition codes on file control cards. Acceptable values are the same as those for Field-6.

Note that use of the values 5, 6, or 7 will prevent collation upon the output file. If one of these values is used and Field-7 is not 1, then this option will override Field-7 and an error message will be produced.

i) Field-9: Borrow Memory Control

The use of this parameter allows the dynamic allocation of free memory to a Sort process at execution time. The free memory area is used for control tables and buffers in addition to the area allocated by the \$ LIMITS card.

The use of this parameter is meaningful only in a LOWLOAD environment. Acceptable parameter values are:

- 0 - No memory is borrowed for the Sort process.
or
null
- n - The n represents a numeric value of 1 to 262,144 memory locations. The Sort process borrows n words of memory (or any available portion of n) at execution time. Memory is borrowed in 1024 word modules. All borrowed memory is released to the operating system following the final collation phase.

j. Field-10: No Option Currently Implemented

This field must be indicated as a null field if subsequent parameters are used.

k. Field-11: Multiple-reel File Control (Optional)

This parameter forces automatic reel switching for unlabeled input files. If used, reel switching is forced at end-of-reel for both Sort and Merge input files. The end-of-input can be indicated only through operator intervention at reel switching time. Acceptable parameters are:

- 0 - Assume normal processing.
or
null
- 1 - Assume input is on multireel unlabeled files.

l. Field-12: Collation Phase Statistic Control

This parameter may be used to request collation file utilization accounting from a sorting process. The statistics include record counts, block counts, and string distributions. Acceptable parameter values are:

- 0 - Do not print statistics.
or
null
- 1 - Print collation statistics.

15. A Special-Names paragraph may consist of a combination of statements shown in Options 1, 2, 3, 4, 5, 8, 9, and either Option 6 or 7.

INPUT-OUTPUT SECTION.
FILE-CONTROL.

FUNCTION: To identify each file referenced in the program, assigning a file-code to each.

Option 1:

FILE-CONTROL. SELECT [OPTIONAL] [OVERLAY] file-name-1 [RENAMING file-name-2]
ASSIGN TO file-code-1 [FOR { CARDS
LISTING
MULTIPLE REEL }]
[, RESERVE { integer-1
NO } ALTERNATE [AREA
AREAS] [FOR BLANK COMMON]] . [SELECT...

Option 2:

FILE-CONTROL. SELECT file-name-1 ASSIGN TO file-code-1 [, file-code-2..].
[SELECT... .]

Notes:

1. As the above format shows, each file is selected in a separate sentence. SELECT sentences of the formats shown in Options 1 and 2 may be freely mixed in the FILE-CONTROL paragraph. The clauses must appear in the order shown in the respective formats.
2. Each file referenced in the program must be selected exactly once in the FILE-CONTROL paragraph. The name of each selected file must be unique within the program.
3. If the RENAMING option is used, the compiler automatically applies the data description of file-name-2 to file-name-1. This includes the File Description entry and the associated Record Descriptions. The SELECT statement for file-name-2 must not in turn contain the RENAMING option; furthermore, file-name-2 must not have a sort-file description. Since RENAMING implicitly supplies the description of file-name-1, the latter must not be explicitly described in the DATA DIVISION. RENAMING does not imply SAME AREA (see I-O-CONTROL).
4. If the RENAMING option is not used, file-name-1 must be described in the DATA DIVISION File Section.
5. The SELECT sentence format shown in Option 1 must be used to SELECT all files other than sort-files (whose data descriptions begin with SD entries). The Option 2 format must be used to SELECT all sort files.

6. An input file which will not necessarily be present every time the object-program is run must be designated OPTIONAL in its SELECT sentence.
7. The OVERLAY option is provided to allow common files in an overlay segmented environment to be opened in an overlay segment and left open to be referenced by subsequent overlays when operating from a production library. This option must be specified in the SELECT statement of all overlay segments which influence the common file except the segment which contains the OPEN statement for the file. Any file specified as an OVERLAY file will not generate a normal File Control Block. Instead, the "Locsym" for the file will be positioned within the correct Labeled Common using BSS pseudo-ops.

It is imperative that all file properties, including any RERUN clauses, be identical for files using this feature.

8. Each file-code must be a two-character word consisting of two letters (A, ..., Z) or a letter and a digit (0, ..., 9). Each file-code must be unique with respect to other file-codes specified in the program. The File Control Block and the buffers for each file are assigned to Labeled Common storage using the file code as the Labeled Common name. Thus, it is possible to establish common files simply by assigning the same file code to the file in each segment or overlay segment which references the file. COBOL reserved words must not be used as file-codes. At execution time, an object-program is submitted to GECOS with "file cards" specifying the peripheral device for each file. The file-code in the file card must be the same as that assigned in the source-program. GECOS associates each of the object-program's files with its peripheral device by matching the file-codes. (For further information refer to the GECOS manual.)
9. Files originating on punched cards or destined for cards or printer must be so specified in the Option 1 SELECT statement format. Failure to designate such files properly may lead to unpredictable data format errors at object time. A file intended FOR CARDS or FOR LISTING should have System Standard Format applied in the I-O-CONTROL paragraph.

10. All tapes employed by a sort procedure for a sort-file are called "collation tapes." A sort-file must be assigned one or more file-codes. The sort procedure will utilize the tape units associated with the assigned file-codes. In addition, the sort procedure requests from GECOS as many UTILITY TAPES as can be allocated to it at object-time, up to a maximum of sixteen tapes (counting each file-code specified as one collation tape). The sort procedure requires at least 3 collation tapes at object time, but additional tapes result in improved execution speed. If the GIVING option is specified in the relevant SORT statement, the file-code of the GIVING file is understood to be available to the sort procedure as a collation tape. The SORT/MERGE manual should be consulted for further information on user control of collation tape allocation.
11. The RESERVE clause allows the user to modify the number of input/output buffers allocated by the compiler. The standard storage assignment for a program is two buffer areas, which is implied if RESERVE is not specified. If the RESERVE clause is specified with the integer option, two buffer areas are allocated. The RESERVE clause specified with the NO option results in a single buffer area being allocated to the file.
12. All integers must be positive.
13. The BLANK COMMON option provides the user a means to override the automatic assignment of any buffer areas for a file to Labeled Common storage. If this option is used, the buffer area for the file will be assigned to Blank Common storage. This can be useful when attempting to load large object programs with a limited amount of memory available. In this case, the Blank Common area can be shared with the loader by using the \$ LIMITS card which indicates the amount of Blank Common storage to be shared. If the LOWLOAD option is used, then the maximum size of Blank Common must be specified on the \$ LOWLOAD card. The octal length of Blank Common is generated on the preface page of each GMAP assembly.

The BLANK COMMON option can be specified for independent programs without creating problems. However, caution should be exercised when applying this feature to segments operating in a multi-segment environment in which files using this option may be common to another segment. If a file common to two or more segments needs to be assigned to Blank Common storage, the file must be assigned in each segment where it is common, and each segment must be re-compiled. Furthermore, an identical ordering of files (by SELECT statements in each segment is necessary if more than one file is involved.

I-O-CONTROL.

FUNCTION: To specify special input/output techniques, files which share the same memory areas, multiple file tapes, and rerun points.

I-O-CONTROL. [APPLY PROCESS AREA ON file-name-1 [, file-name-2...]]
 [; APPLY BLOCK SERIAL NUMBER ON file-name-3 [, file-name-4...]]
 [; APPLY { SYSTEM STANDARD } FORMAT ON
 VLR }
 file-name-5 [, file-name-6...]]
 [; RERUN [ON file-name-7] EVERY integer-1 RECORDS OF file-name-8]
 [; RERUN...] [; SAME [RECORD] AREA FOR file-name-9, file-name-10
 [, file-name-11...]] [; SAME...]
 [; MULTIPLE FILE TAPE CONTAINS file-name-12 [POSITION integer-2]
 [, file-name-13 [POSITION integer-3]..]] [; MULTIPLE...].

Notes:

1. This paragraph may be omitted when none of its options is desired.
2. APPLY PROCESS AREA causes a special input/output method to be used on the specified files. In addition to the normal buffer(s), the compiler allocates a logical record area to each file. On an input file, each logical record is implicitly moved to the fixed PROCESS AREA for processing when it is read. On an output file, each logical record is developed in the fixed PROCESS AREA, and implicitly moved to the buffer when it is written. The standard input/output method (for other than PROCESS AREA files) is to process each logical record right in the buffer area. The APPLY PROCESS AREA option can be used to increase object-program efficiency on blocked or buffered files with heavy processing activity.
3. APPLY BLOCK SERIAL NUMBER causes a special input/output error control feature to be used on the specified files. Each physical block has incorporated in it a serial number, which is checked by the input routine when the file is read (in this or another program). If a file has block serial numbers, it must be mentioned in this APPLY clause in all programs that reference it. The use or omission of this option has no effect on sort-files, which always have block serial numbers.

4. The APPLY SYSTEM STANDARD or VLR FORMAT ON statement allows the user to assume explicit control over the data format of a file, provided the description of the file in the Data Division is consistent with the specified format. System Standard Format implies:
 - a) Block serial numbers are to be applied;
 - b) VLR format is to be applied (see below);
 - c) Recording mode must be binary high density, (this mode is always assumed when the RECORDING MODE clause is omitted);
 - d) Overall data block sizes must not exceed 320 words, or 1920 characters: (the compiler assumes the desired block size is 320 words when the BLOCK CONTAINS clause is omitted); and
 - e) Label records must be standard (COBOL rules require this option to be stated explicitly in the FD entry in the Data Division).

VLR (Variable Length Record) format implies that each logical record will be immediately preceded in the buffer by a record control word, which contains the record size, in words, and other control information. Record control words are retained in the external medium except on punch cards and printer listings. VLR format requires binary recording mode. Various other options can cause the compiler to apply VLR format implicitly; the APPLY VLR option may be used explicitly when VLR format is desired but would not be implied by other options specified for the file.

5. The RERUN option causes checkpoint memory dumps to be written. If "ON file-name-7" is specified, the output device allocated to file-name-7 receives the checkpoint dump; otherwise the output device allocated to file-name-8 receives the checkpoint dump. If the ON option is used, file-name-8 may be either an input or an output file. Integer-1 must not exceed 250,000. It is the user's responsibility to keep the output device in a suitable status to receive the checkpoint dump (i.e., opened as an output file) at every point in the program where a READ or WRITE references file-name-8.

When specified for a sort-file, the RERUN option indicates that checkpoints are to be taken during the Sort. In this case integer-1 is ignored. The Sort determines when checkpoints have meaning and should be taken. The ELECT SORT OPTIONS under SPECIAL-NAMES can also be used to specify that checkpoints are to be taken during a Sort.

6. The files mentioned in a SAME clause share memory areas. If the RECORD option is used, the PROCESS AREA feature is automatically applied to the files mentioned, and they share only the fixed PROCESS AREA. If several files with the SAME RECORD AREA are open concurrently, the logical record of only one of the files can exist in the record area at one time. If the RECORD option is not used, the files share the same buffer area(s). The user must assure that no conflict results from having two or more SAME AREA files open concurrently; the results of such a conflict are unpredictable. A given file may be mentioned in only one SAME clause; several SAME clauses may be used if desired. A sort-file must not be mentioned in a SAME clause.

7. The MULTIPLE FILE option is required if two or more files share the same reel of tape. Of the files on such a tape, only those referenced elsewhere in the source-program need be specified. If all file-names on the tape are listed consecutively, POSITION may be omitted. If any file in the sequence is not listed, the POSITION relative to the beginning of the tape must be given. The file-name of a sort-file must not appear in a MULTIPLE FILE clause. Either all labels must be present or else all labels must be omitted for the files on a MULTIPLE FILE tape. Each MULTIPLE FILE clause describes one MULTIPLE FILE tape. There can be any number of MULTIPLE FILE input or output tapes; however, all files listed for each tape must be contained on a single reel. OPTIONAL files are not permitted on MULTIPLE FILE tapes. Note that files on a MULTIPLE FILE tape cannot be open concurrently.

IX. IDENTIFICATION DIVISION

GENERAL DESCRIPTION

The purpose of the IDENTIFICATION DIVISION is to identify the Source Program and outputs of a compilation. In addition, the user may include the date that the program was written, the date that the compilation of the Source Program was accomplished, and any other information which is desired.

ORGANIZATION

Fixed paragraph names are used as keys in this division. They identify the type of information contained in the paragraph.

The name of the program must be given in the first paragraph. This paragraph is named PROGRAM-ID. Other paragraphs which may be included in this division are:

AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
REMARKS.

PROGRAM-ID

FUNCTION: To give the name by which a program is identified.

PROGRAM-ID. program-name.

Notes:

1. The program-name is a word and must conform with the rules for a word. It must consist of one to six characters, with at least one letter and the remaining characters letters and/or digits.
2. The program-name should be used to identify the Source Program, the Object Program, and all listings pertaining to a particular program.

DATE-COMPILED

FUNCTION: To provide the compilation date in the IDENTIFICATION DIVISION.

[DATE-COMPILED. any sentence.]

Notes:

1. The paragraph-name DATE-COMPILED causes the current date to be inserted during program compilation. If a DATE-COMPILED paragraph is present, it is replaced during compilation with a paragraph of the form:

DATE-COMPILED. current-date

X. SOURCE-PROGRAM SEGMENTATION

INTRODUCTION

The objectives of the source-program segmentation provisions discussed below are:

1. To permit practical separation of a data-processing program into distinct functional modules ("segments").
2. To permit the segments to be developed as separate COBOL source-programs which are compiled separately and may be debugged separately.
3. To permit linking of the segments of a program by the object-program loader.
4. To permit the functional modules ("segments") to overlay other modules when called into memory in order to execute large programs within a limited amount of memory.

Four distinct communication problems arise in segmenting data processing programs. The first is communication of information contained in the data file buffers and housekeeping information which is common between two or more segments. The second is communication of working storage data common to two or more segments. The third is communication of constant storage data common to two or more segments. The fourth is communication of procedural control. Other problems arise when a data processing problem is segmented to function in an overlay environment. One problem that must be considered when operating in an overlay environment is how to control files which are common to the two or more segments. There should be a communication ability which allows loading of an overlay segment which restores any common areas to their initial states or allows them to remain in their current states.

As a special feature, the GE-635 COBOL compiler provides solutions to all four communications problems, as described below. It should be understood that source program segmentation (as specified for GE-635 COBOL which utilizes the source program segmentation features) must generally be rearranged into a unified, non-segmented form if it is to be subsequently compiled on a different computer line. Such rearrangement is not necessary for non-segmented GE-635 COBOL source-programs.

SEGMENTS

Segments are subprograms which are compiled and tested independently and subsequently loaded together and executed as a total program. Thus, a user may decide to break up a large complex program into several parts or segments, write each one as a separate source program, and compile and test each segment independently; thereby overlapping programming and checkout time. Another use of segments is to facilitate the writing of common subroutines (installation oriented) in source language to be compiled as independent segments.

SECTIONS

Sections consist of a section header followed by one or more successive paragraphs. They generally contain a common function which is executed from more than one place in a program. The programmer is free to partition a program into sections as he chooses. A section ends immediately before the next section-name or at the end of the Procedure Division or, in the Declaratives Section of the Procedure Division, at the key words END DECLARATIVES.

The GE-635 COBOL compiler does not provide segmentation of COBOL procedural sections as specified by COBOL-65 standards. However, it is possible to functionally organize a group of 600 COBOL segments to operate in the same manner as segmentation of COBOL-65 procedural statements, with the exception that there is no way to call a segment and have it made available in its last-used state.

A discussion of the four communication methods is described below:

DATA FILE COMMUNICATIONS

The GE-635 COBOL compiler automatically places the file control blocks and all buffers for each file named in the File Section into Labeled Common storage. The name of the Labeled Common storage area is assigned using the two-character file code specified in the ASSIGN clause for the file. At load time, the loader will allocate all Labeled Common storage areas having the same name to the same area of memory. Therefore, if a file is referenced by two or more segments, it must be described identically in the Data and Environment Division of all source programs referencing it. There may be times when it is desired to allocate the file buffers to Blank Common storage rather than to Labeled Common storage. This can be accomplished by using the FOR BLANK COMMON clause specified in the FILE-CONTROL paragraph. This can be very useful if it is desired to allow the loader to share the buffer areas of the file at load time, to decrease the memory requirements for job allocation. For details on loading Blank Common versus Labeled Common, refer to the manual GE-625/635 General Loader, CPB-1008.

A report can be referenced only from within the segment in which it is described. If several reports going to a single file are to be generated in separate segments, each report must be specified in the segment which contains the relevant Report Writer verbs, and may be omitted from segments not containing relevant Report Writer verbs. The complete file description, excluding unreferenced report descriptions, must appear identically in each reporting segment.

Files sharing the SAME AREA or SAME RECORD AREA must all be described in any segment referencing any of them. Files which are common to two or more segments may be defined to share buffer areas with other files which are common to the same segments. However, when indicating this, the SAME AREA clause must be identical for all segments which share the common files.

USE procedures are linked to one Input/Output file (File Control Block) at the time a file is opened. If applicable USE procedures have been specified within the segment opening the file, they are engaged; any other USE procedures specific for the same common file are not meaningful at this time. Therefore, USE procedures for a specific file should be present only in segments in which the file is opened. To place USE procedures in other segments which share the common file wastes memory space. When a file is common to more than one segment, it is permissible to reference the file from any of the segments in which it is common. However, if not all READ or RETURN verbs referencing a file appear in the same segment, then each such verb referencing the file must have an explicit AT END procedure.

WORKING-STORAGE COMMUNICATIONS

When designing a system to be modular to facilitate development and checkout, it is not uncommon to want to define particular items or records in Working-Storage such that they are common to all segments when the segments are operating together. However, it is also necessary that storage be assigned such that each segment can be developed independent of other segments. GE-635 COBOL provides a feature which allows the user to define Working-Storage items to be assigned to Labeled Common storage in as many segments as desired. Items not explicitly allocated to Labeled Common storage are allocated space in the segment itself. Working-Storage items may be assigned to Labeled Common storage via the BLOCK clause option of the SPECIAL-NAMES paragraph. Any data item which is a level 01 or 77 may be assigned to Labeled Common storage. Labeled Common storage areas can be made equivalent simply by specifying the same label-name for corresponding BLOCK clauses in segments.

When segments reference Working-Storage items which are common to other segments, it is advisable to define the area in each segment using identical names and data descriptions. Each segment must contain a BLOCK statement which identifies the Labeled Common area using the same label-name. It is permissible to assign initial values for items which are described in labeled storage areas. The user should be aware that the loader re-establishes the labeled storage area when a segment is loaded. Therefore, when loading a production run which is segmented, the last segment which defines the labeled common area, using initial values, will determine the values of the items when the program begins execution.

CONSTANT COMMUNICATION

The Constant Section consists of many items containing initial values and which are likely common to many segments when a large data processing problem is designed using the modular concept. GE-635 COBOL allows any level 01 or 77 item defined in the Constant Section to be assigned to Labeled Common storage. The user can assign the entire Constant Section into one labeled storage area by using the THRU option of the BLOCK clause in the SPECIAL-NAMES paragraph. When defining constants for a common storage area the descriptions for all items should be identical in each segment.

A source program may include as many BLOCK clauses as are needed. The total number of file codes and BLOCK labels must not exceed 63 for a given segment.

PROCEDURAL COMMUNICATIONS

COBOL requires a PROGRAM-ID to be specified for each source program. In 600 COBOL the PROGRAM-ID must be one to six characters in size and it must be comprised of letters and/or digits, including at least one letter.

The compiler uses the PROGRAM-ID for the implied entry symbol for identification of the COBOL object-program. The implied entry point for a source program is the first procedural statement following the END DECLARATIVES statement if declaratives are used, or the first statement of the Procedure Division if declaratives are not used. The entry symbol (the PROGRAM-ID) is made a global symbol which allows referencing by calls from other segments.

Each 600 COBOL object-program has the basic structure of a single-entry closed subroutine. This is true even though any number of entry points may be defined within each segment. The compiler automatically generates the entry linkage coding at the entry point (the PROGRAM-ID), and the implied exit linkage coding is generated at the end of the last procedural statement of the program. Coding is generated at each entry point and exit which will save and restore all index registers when an entry is called as a closed subroutine. Inter-communication between 600 COBOL object-program segments does not require index register integrity to be maintained. The Save and Restore index register feature is implemented in 600 COBOL to allow communication between COBOL segments and non-COBOL segments when operating in a segmented environment. If a segment has only one entry point (the implied entry) and it is to be executed as a closed subroutine by another segment, the program should be arranged so as to "fall through" to the exit linkage at the appropriate time. An EXIT paragraph may be used for this purpose if necessary.

CALL Verb

The CALL verb provides a way to transfer control to a separately compiled subprogram or entry point within a subprogram, with a standard return mechanism provided. An independently compiled COBOL program may be called as a closed subroutine by a CALL using the PROGRAM-ID which names the segment. If it is desired to execute only a portion of a subprogram from another subprogram, the CALL should reference one of the entry-names defined in an ENTRY POINT statement. A COBOL CALL can reference non-COBOL subprograms provided that the called name has been established as a global symbol within the called segment.

For each CALL statement, USING arguments can be specified to provide address pointers to data which is to be used by the entry-name being called. The USING arguments are not meaningful if the CALL references the PROGRAM-ID of a 600 COBOL subprogram. The arguments provide indirect pointers to "input" and "output" data fields when a CALL references an entry-name which is defined using the ENTRY POINT Statement and has USING and GIVING arguments included. Data-names specified as USING arguments must be level 77 or 01 items defined in the Working-Storage or Constant Section of the subprogram in which the CALL is defined.

The number of USING arguments specified with a CALL statement must correspond exactly with the number of USING and GIVING arguments defined for the entry-name which the CALL references. The data description for each pair of corresponding arguments must be identical when a CALL references a COBOL subprogram. If the CALL references a non-COBOL subprogram the data descriptions should provide a data format which is compatible with the called subprogram.

ENTRY POINT

600 COBOL provides the ENTRY POINT feature for the definition of entry points in addition to the implied entry point. It is possible to organize a program such that paragraphs, sections, or combinations of paragraphs and sections can be called and executed from other segments. Each entry-name must be unique and must not contain more than six characters with at least the first two characters being letters and the remaining characters defined as letters and/or digits. An ENTRY POINT can be used any place in the Procedure Division except in the Declarative Section. Entry-names can be referenced only through calls from other segments. They must not be referenced by a CALL from within the segment in which they reside. The compiler generates one 9-word save area in each segment for the preservation of index registers and indicators. Therefore, a segment which has been called may itself contain CALL statements. However, a called segment or entry point must not contain a CALL statement that directly or indirectly calls the calling program.

EXIT Verb

The compiler provides an EXIT entry-name option which defines an unconditional exit for a given entry-name. Each entry-name (explicit) in a segment must have at least one exit defined using the EXIT entry-name statement. Multiple exits may be defined for an entry-name if necessary. If control reaches an EXIT entry-name statement, the linkage control stack is checked to determine if the current EXIT corresponds to the called entry-name. If the EXIT corresponds to the entry-name, the EXIT causes control to return to the calling program immediately following the CALL statement. If it does not correspond to the entry-name, control passes through the exit point to the first sentence of the next paragraph. An entry-name with its associated EXIT and/or EXITS can be nested within other entry-names and their associated EXITS. ENTRY POINTS and EXITS may be placed such that they extend across other entry-names and/or exits. Since the same push-down stack is used for entry-names as is used in performing paragraphs, extreme care should be given to provide the orderly push-down and popping-up of the control stack when performing paragraphs within coding for an entry-name. The EXIT PROGRAM option is implemented to provide an unconditional exit from a segment when operating under the control of a CALL statement. This feature causes control to return to the point in the calling program immediately following the CALL statement. If control reaches an EXIT PROGRAM statement and no CALL statement is active, control passes through the EXIT point to the first sentence of the next paragraph.

DATA COMPATIBILITY

600 COBOL provides excellent facilities for the processing of common data between COBOL segments. The fact that all files are assigned to Labeled Common storage and the ability to assign any or all items in working storage to Labeled Common provides complete flexibility for referencing common data when only 600 COBOL segments are involved. An additional feature is provided to allow the communication of data between non-COBOL segments and COBOL entry points, other than the implied entry, by means of arguments.

USING and GIVING arguments may be specified with the ENTRY POINT statement. Any USING arguments associated with an entry-name reference items within the segment which function as receiving fields for the move of input arguments which take place when the entry-name is called from another segment. The compiler assumes the external format of the item is compatible with the COBOL description specified for the item in the Data Division. The user must assure that the order and descriptions of the arguments conform to the calling programs USING argument list. No more than 10 USING and GIVING arguments may be defined with an ENTRY POINT statement. USING argument data-names must reference level 77 or 01 items defined in the Working-Storage or Constant Sections. The procedural statements following an ENTRY POINT statement are not executed until individual moves of the USING argument list are complete.

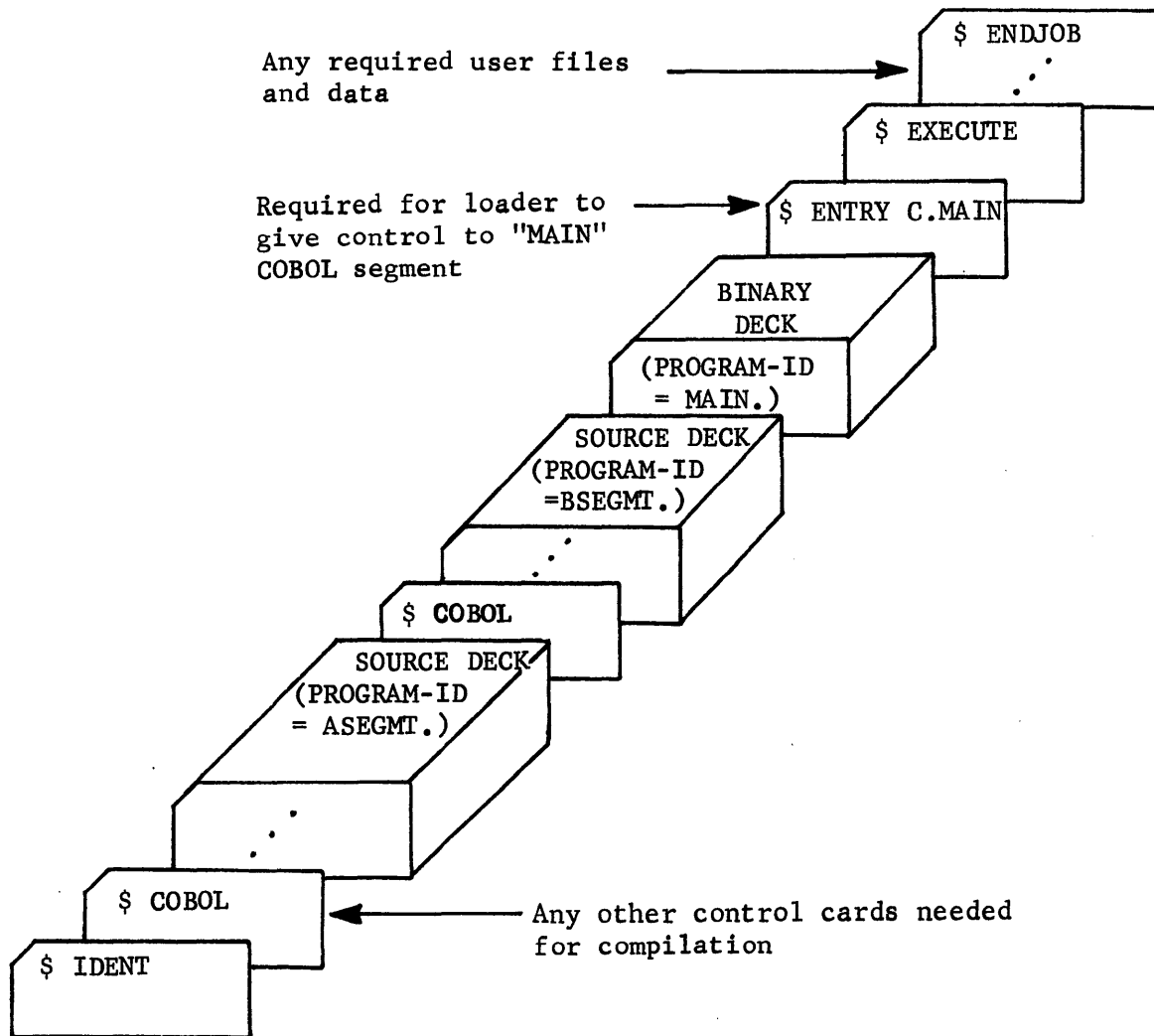
The GIVING arguments used with an entry-name reference items in the Working-Storage or Constant Section which function as sending fields for moves of output arguments which take place when an EXIT entry-name is recognized. The compiler assumes the desired external format is compatible with the COBOL descriptions specified for the item. GIVING argument data-names must reference level 77 or 01 items defined in the Working-Storage or Constant Sections. Control is not returned to a calling program when a valid EXIT is detected until individual moves of the GIVING argument list is completed. The GIVING moves use the corresponding argument address associated with the controlling CALL statement from the calling program as a receiving field. The user must assure a one-to-one correspondence between the Call-USING arguments and the ENTRY POINT USING/GIVING arguments.

Item format descriptions within 600 COBOL which may be compatible with external formats of non-COBOL segments are DISPLAY, COMPUTATIONAL-1, COMPUTATIONAL-2, and COMPUTATIONAL-3. The COMPUTATIONAL format of 600 COBOL should not be used to describe USING or GIVING items. If it is desired to maintain a high degree of decimal precision during computations involving the USING data-names, the appropriate conversion can be accomplished by a move of the item to a field defined appropriately as COMPUTATIONAL.

EXAMPLE OF MULTI-SEGMENT JOB

An example of a three-segment COBOL job is given on the next page. The example supposes that the main segment (PROGRAM-ID is MAIN) has already been compiled and that two other segments must be recompiled along with an execution activity of the three segments. Since the loader would give control to the first primary SYMDEF encountered in the deck setup, it is necessary in this example to use a \$ ENTRY card in order to force control to be given to the main segment. COBOL conventions provide a primary SYMDEF for this purpose. It is a six-character symbol in which the first two characters are always "C." and the remaining four characters are taken from the first four characters of the PROGRAM-ID.

Example Deck Setup For Multi-Segment Compile And Go.



APPENDIX A
COBOL COMPUTATIONAL ITEM FORMATS

This appendix provides detailed information on the machine formats for COMPUTATIONAL[-n] items. The COMPUTATIONAL (without suffix) USAGE is recommended for general use. The other options for the USAGE clause may be employed for special purposes.

The functions of the various COMPUTATIONAL options are given below:

COMPUTATIONAL - Results in the decimal precision format. This is the preferred USAGE for items referenced in calculations.

COMPUTATIONAL-1 - Results in the fixed point binary integer format. This USAGE is well suited to items having only integral values, and is the preferred USAGE for items used as subscripts or referenced in DEPENDING options.

COMPUTATIONAL-2 - Results in the floating point binary format. This USAGE is appropriate for items whose absolute values may potentially exceed 10^{18} or be less than 10^{-18} ; it is also useful for data communications with non-COBOL programs.

COMPUTATIONAL-3 - Results in the single precision fixed point binary integer format. This USAGE should be employed only for data communications with other non-COBOL programs; even then COMPUTATIONAL-1 or COMPUTATIONAL-2 should be used instead if the application permits.

Representation Of Fractional Values

The most important feature of the COMPUTATIONAL USAGE is a special provision for the fractional part of an item. For a COMPUTATIONAL item, the fractional part and integral part are jointly represented as a binary integer, which corresponds exactly to the conceptual decimal value of the item. For a COMPUTATIONAL-2 item, however, the fractional part of the value is represented as a pure binary fraction of a limited (finite) number of bits. (Both COMPUTATIONAL-1 and COMPUTATIONAL-3 items are integers, and therefore have no fractional parts.)

Generally speaking, a decimal fraction of a given number of digits cannot be represented exactly by a binary fraction of any finite number of bits. Consider for example, the value $1/5$, which is represented in decimal notation as 0.2. Trying to represent it by a 4-bit binary fraction, one obtains $(.0011)_2$ or $3/16$; with 8 bits, one obtains $(.00110011)_2$ or $51/256$. In fact, the exact value must be written as

$$(0.2)_{10} = (0.\overline{0011})_2$$

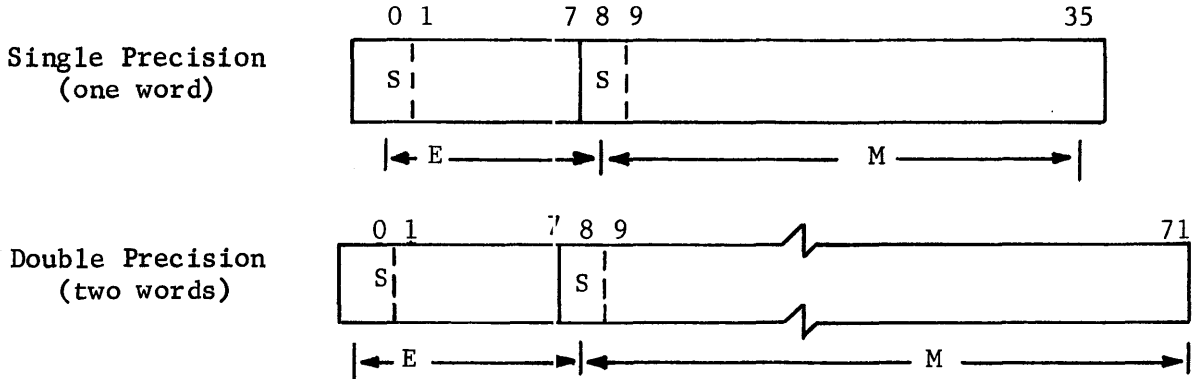
which means that the bit pattern 0011 in the binary expansion keeps repeating indefinitely. If the decimal value 0.2 is converted to a binary expansion of, say, 71 bits, and then converted back, the 1-digit result would be 0.1, quite different from 0.2. (The 4-digit result would be 0.1999, which is almost-but not quite-equal to 0.2.) If computations were involved instead of only conversions, the imprecision in the decimal result could be much greater.

Various adjustments can be made to binary fractional values to make exact decimal results highly probable. The sure way, however, is to use binary integer notation to represent all values, whether integral or fractional. A consequence of doing so is that multiplication or division of an operand by a power of ten is sometimes necessary in the course of a computation. COMPUTATIONAL items indeed do use binary integer multiplications or divisions by powers of ten. (The formats and conventions governing COMPUTATIONAL items are fully described on the following page.)

In most commercial data processing applications, particularly where dollars and cents are involved, a high degree of decimal precision is expected. It is for this reason that COMPUTATIONAL and COMPUTATIONAL-1 usages are recommended over COMPUTATIONAL-2 and COMPUTATIONAL-3.

COMPUTATIONAL Items

The basic machine format for COMPUTATIONAL items is the standard floating point binary format (single or double precision):



The decimal precision format utilizes the above format in a special way. The exponent (E) indicates how many bits of significant information are present in the mantissa (M). Bits to the right of the point indicated by the exponent are not significant; these bits are normally all zero.

The sign (S) of the exponent is normally nonnegative. The sign (S) of the mantissa is the algebraic sign of the COMPUTATIONAL item.

The value stored in the mantissa is a binary integer, obtained as follows: If the item's data description specifies fractional places, the mantissa is stored as if the value had been multiplied by a sufficiently high power of 10 to make the value an integer. (The power of ten is called the "span multiplier.") Thus 3.142 would be stored as 3142, as if it had been multiplied by 10³. (The actual value in memory, expressed in octal notation would be 030610600000.)

In general, if an item's PICTURE is 9(p)V9(q), a suitable span multiplier is 10^q or any higher power of 10. If the span multiplier actually chosen is 10^s (with $s \geq q$), the s is called the "span number." The "span" of a Decimal Precision number is defined as the number of fractional digits permitted for an item in view of its span multiplier. Referring to the above example, 3.142 would be assigned "Span 3", allowing 3 fractional digits.

The significance of the conventions just described is that a binary fraction or mixed number "equivalent" to the decimal value could in general only be approximate, not exact, but the span multiplier permits the value stored to be exact in the Decimal Precision format.

The compiler of course chooses the span multiplier for each COMPUTATIONAL item, and supplies appropriate coding to align the operands and results in all computations. A "span conversion" is sometimes required for this purpose; that means multiplication or division by a suitable power of 10 (always with a positive exponent). General rules for COMPUTATIONAL item alignment are as follows:

1. If a MOVE statement, or an addition or subtraction function, involves operands with the same span number, they are properly aligned without span conversion. Otherwise one or more span conversions will be necessary.
2. The span number of a product equals the sum of the span numbers of the operands, so span conversion via division is usually necessary in order to cause proper alignment of the result.
3. The span number of a quotient equals the difference of the span number of the dividend and that of the divisor, so span conversion via multiplication is usually necessary in order to cause proper alignment of the result.

Because of the first rule above, it is desirable to have summands in the same span, to avoid span conversions. Here another Decimal Precision format convention becomes important: Since a given item's span number can in principle be any number equal to or greater than the number of fractional places in the item's description, items with widely different PICTURES can often be assigned the same span.

For example, Span 3 in single word precision can permit 0 to 3 fractional places and 0 to 5 integral places. In double word precision, Span 3 can permit 0 to 3 fractional places and 6 to 15 integral places. Since items with no more than 3 fractional places are extremely common in commercial data processing applications, it is clearly desirable to assign Span 3 to items whenever possible, even if they have only 1 or 2 fractional places, to optimize their formats for addition and subtraction. This reasoning leads naturally to the concept of "preferred" spans.

Certain spans, each of which is applicable to a wide range of PICTUREs, are "preferred," in the sense that an item is always assigned to a preferred span if its PICTURE permits. Furthermore an order of preference is applied; thus an item which could be assigned either to the span which is "preferred #1" or to that which is "preferred #2" would be assigned to the former. When the span has been assigned, it can be determined whether the item's PICTURE requires single or double precision, with single precision chosen whenever possible. The following tables fully describe the rules for span and precision assignments:

Single Precision Items

Integral Places	Fractional Places	Span Number	Comment
1-8	0	0	Preferred #1
0-5	1-3	3	Preferred #2
0-3	4-5	5	Preferred #3
0	8	8	Preferred #4

Double Precision Items

Integral Places	Fractional Places	Span Number	Comment
9-18	0	0	Preferred #1
16-17	1	1	
16	2	2	
6-15	1-3	3	Preferred #2
14	4	4	
4-13	4-5	5	Preferred #3
11-12	6	6	
11	7	7	
1-10	8	} 8	Preferred #4
0-10	6-7		
7-9	9	9	
7-8	10	10	
7	11	11	
0-6	9-12	12	Preferred #5
4-5	13	13	
4	14	14	
0-3	13-15	15	Preferred #6
1-2	16	16	
1	17	17	
0	16-18	18	Preferred #7

For optimal results, it is recommended that the items in a program occur mostly in Span 0 and Span 3, and that single precision be used whenever possible. As the tables show, most practical PICTUREs specifying eight or fewer digits will result in single precision formats. Referring to the tables, the user can force the choice of a given span and precision by supplying an appropriate PICTURE.

The following examples illustrate the results of various span combinations:

1. Suppose A has PICTURE 9V99 and B has PICTURE 9(5)V9(3), and their sum must be computed. The tables show that both are single precision Span 3 items, so no span conversion is necessary.

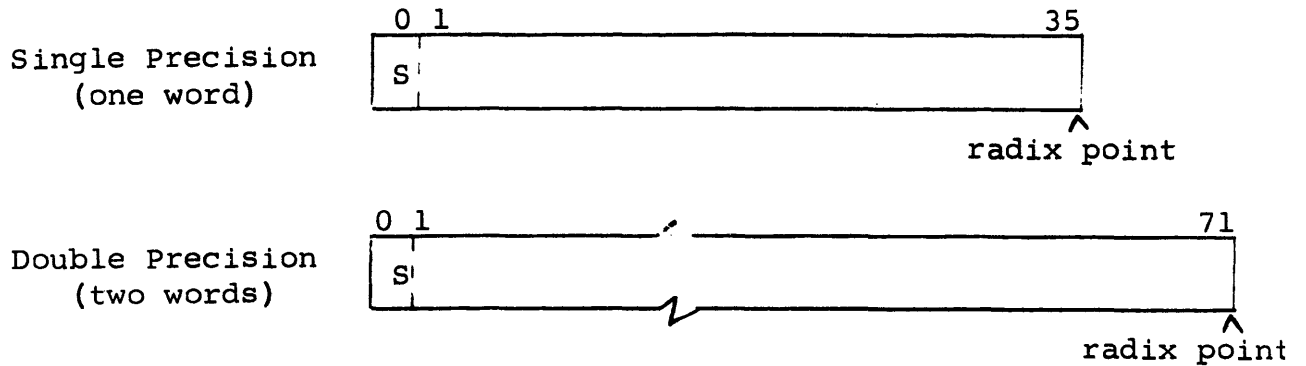
2. Suppose C has PICTURE 9(6) and D has PICTURE 9(6)V99. Then C is a single precision Span 0 item and D is a double precision Span 3 item. If their sum is desired, a span conversion will be necessary. If their product, however, is to be a double precision Span 3 number, no span conversion is required for the multiplication.
3. Suppose E has PICTURE 9(3)V9(3) and F has PICTURE 9(10)V9(3). Then both are assigned Span 3, but E is single precision and F double. They may be added without conversion to produce a Span 3 sum. Suppose, however, the product is to be stored in a Span 3 item. The product of two Span 3 numbers is a Span 6 number. Therefore, a span conversion from Span 6 to Span 3 must follow the multiplication. Specifically, the Span 6 product must be divided by 10^3 . If a Span 3 quotient is desired, division of two Span 3 numbers results in a Span 3 quotient which must be converted to a Span 3 via multiplication by 10^3 .

The use of machine floating point format for COMPUTATIONAL items permits many operand alignment steps to occur automatically via floating point hardware, thus saving space and time in the object-program. Another advantage is that single and double precision operands can be mixed arbitrarily in a floating point computation without the need for programmed conversions. A computation proceeds in double precision only when at least one of the operands is a double precision item.

Still another important advantage of using floating point format is that in a computation involving several arithmetic operations (resulting from a complicated COBOL formula, for example), the hardware retains extra significant information in each intermediate step so that the conceptual 18-digit limit on operands may sometimes be meaningfully exceeded on intermediate results. The overall significant figure, however, never exceeds 21 digits (so that the result of multiplying two 18-digit numbers, for example, cannot be 36 digits even in an intermediate result). The 18-digit limit always applies to stored values.

COMPUTATIONAL-1 Items

The basic machine format for COMPUTATIONAL-1 items is single or double precision Fixed Point Binary Integer format:



Although it is stored as a binary number, a COMPUTATIONAL-1 item's value is exactly equivalent to the conceptual decimal value of the item. This is true because COMPUTATIONAL-1 items are restricted to integral values; every decimal integer has an exact equivalent in a binary integer.

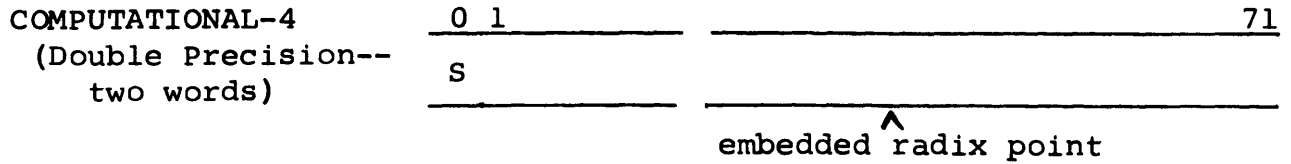
The following table provides the precision assignment rules for COMPUTATIONAL-1 items:

<u>Number of Digits</u>	<u>Precision</u>
1-8	Single
9-18	Double

For efficiency, COMPUTATIONAL-1 items should have 8 or fewer digits whenever the application permits.

COMPUTATIONAL-2 Items

The basic machine format for COMPUTATIONAL-2 items is the single or double precision Floating Point Binary format:



Sometimes a program coded in assembly language requires the Fixed Point Binary format for an item. COMPUTATIONAL-3 and COMPUTATIONAL-4 USAGES are intended only to permit data communications with such a program. The reasons are discussed under Representation of Fractional Values, above.

When a Fixed Point Binary mixed number must be used, it is usually best to place the assumed (embedded) radix point as far to the left as possible, to allow as much room as possible for the binary fractional part. This means, of course, that the integral part should be allotted as few bits as the PICTURE allows. An item formatted in this way is said to have "minimum binary scale." (The "binary scale" is the number of integral bits in the item's format, in view of the assumed radix point position.)

The BITS option of the POINT LOCATION clause may be used to specify the exact position of the assumed radix point. In this option, one specifies the desired number of fractional bit positions, counting from the right end of the item. In terms of the COBOL clause format, which is

POINT LOCATION IS LEFT integer BITS

the following relationships determine the item's binary scale:

Single Precision: binary scale = 35 - integer.

Double Precision: binary scale = 71 - integer.

Unless the BITS option is specified, a standard binary scale is assumed by the compiler, as determined by the following table. The cross-hatched areas are inapplicable, since COMPUTATIONAL-3 items may have at most 10 digits specified, while COMPUTATIONAL-4 items may have up to 18 digits.

Minimum Binary Scale

Number of Decimal Integer Places in PICTURE	Standard Binary Scale		POINT LOCATION Equivalent (BITS LEFT)	
	COMPUTATIONAL-3	COMPUTATIONAL-4	COMPUTATIONAL-3	COMPUTATIONAL-4
0	1	1	34	70
1	4	4	31	67
2	7	7	28	64
3	10	10	25	61
4	14	14	21	57
5	17	17	18	54
6	20	20	15	51
7	24	24	11	47
8	27	27	8	44
9	30	30	5	41
10	34	34	1	37
11		37		34
12		40		31
13		44		27
14		47		24
15		50		21
16		54		17
17		57		14
18		60		11

The Data Division statements pertaining to files appear in the File Section:

```

FD file-name  [; RECORDING MODE IS ( BINARY ) [ ( HIGH ) DENSITY ] ]
              [; FILE CONTAINS ABOUT integer-1 RECORDS ]
              [; BLOCK CONTAINS [integer-2 TO ] integer-3 ( RECORDS
              CHARACTERS ) ]
              [; RECORD CONTAINS [integer-4 TO ] integer-5 CHARACTERS ]
              ; LABEL ( RECORDS ARE ) ( STANDARD
              RECORD IS ) ( OMITTED )
              [; VALUE OF data-name-1 IS literal [ ,data-name-2 IS.. ] ]
              { { ; DATA ( RECORD IS
              RECORDS ARE ) } data-name-3 , data-name-4... }
              { { ; REPORT IS
              REPORTS ARE ) }
              ; DATA ( RECORD IS
              RECORDS ARE ) ... ; ( REPORT IS
              REPORTS ARE ) ... }
              [; SEQUENCED ON data-name-5 [ , data-name-6... ] ].
  
```

Of the above clauses, FILE CONTAINS, RECORD CONTAINS, and SEQUENCED have no effect on file formats or processing, but may be used to provide valuable documentation about the file.

EXTERNAL FORMAT CONSIDERATIONS

A file's "external format" is its manner of representation in a peripheral storage medium.

Certain general considerations pertain to the overall file:

1. The actual peripheral medium;
2. The multiple file option (applicable only to magnetic tape files);
3. Recording mode (which on magnetic tape may be binary or BCD, with high or low density);

4. Presence or absence of standard label records.

Other considerations pertain to the contents of each data block (physical record) of the file:

1. Presence or absence of block serial number.
2. Blocking factor (number of logical records per block) or block size (number of data characters or computer words per block).
3. Logical record format--FLR (fixed-length records) or VLR (variable-length records).

Logical Record Format

In GE-625/635 COBOL, each logical record in a file begins in the first character position of a computer word. For example, a logical record consisting of a single 21-character elementary item would be stored as follows:

d	d	d	d	d	d	first word
d	d	d	d	d	d	
d	d	d	d	d	d	
d	d	d	x	x	x	last word

In this example, the character positions represented by d contain data, while those represented by x are unused. Unused positions resulting from this convention, as well as those resulting from USAGE or from SYNCHRONIZED items, appear as shown both in the computer memory and on the peripheral storage medium.

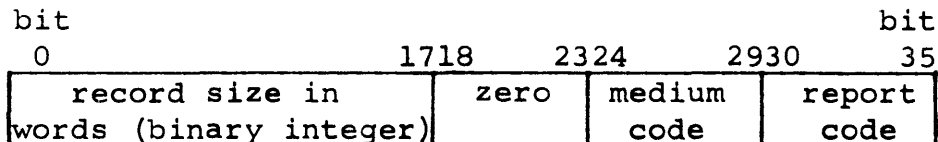
When all of the logical records of a file require the same number of computer words, the FLR format may be used. In the FLR format, the uniform record size is shown from the detailed record description. In a block of several FLR records, the successive records are adjacent to each other, with no intervening control information. A FLR block comprised of records similar to that shown above would appear as follows:

d	d	d	d	d	d	} first record
d	d	d	d	d	d	
d	d	d	x	x	x	
d	d	d	d	d	d	} second record
d	d	d	d	d	d	
d	d	d	d	d	d	
d	d	d	x	x	x	
		--				} etc.
		--				
		--				

If the data records in a file may have different sizes (in computer words), the VLR format is required. Typical circumstances requiring VLR format are:

1. Two or more data records of unequal sizes (in computer words) have been described for the file. In this case, the size of each record type is fixed, but the record size may vary from one record to the next.
2. The OCCURS...DEPENDING option appears in the description of one or more of the data records of the file. The size of an OCCURS...DEPENDING table varies from one record to the next, causing the overall record size to vary, even if only one data record type has been specified for the file.
3. The file is to receive one or more reports generated via Report Writer.
4. The file is to have System Standard format (see System Standard Format, below).

On the peripheral medium (and in the input/output buffers in memory), each logical record in a VLR file is immediately preceded by a "record control word". The record control word is supplied and interpreted automatically by the input/output housekeeping routines, and is not accessible to the object-program. The record control word occupies one computer word, and has the following format:



The record control word is not considered to be a part of the logical record, and therefore the record size subfield does not count the control word itself.

The medium code is determined as follows:

- 2 - FOR CARDS is specified in the SELECT sentence for the file.
- 3 - Either this record is a report line generated via Report Writer, or FOR LISTING is specified in the SELECT sentence, or both.

0 The medium code is zero except in the circumstances just stated.

The report code is normally zero. Via Report Writer, however, the user may control the contents of this subfield. He might wish to produce, say, four reports on the same output file, for later printing. The report code specification requires two steps:

1. In SPECIAL-NAMES, specific one-character code values must be associated with mnemonic names.

SPECIAL-NAMES. "1" IS CODE-1,
 "2" IS CODE-2, "3" IS CODE-3,
 "4" IS CODE-4.

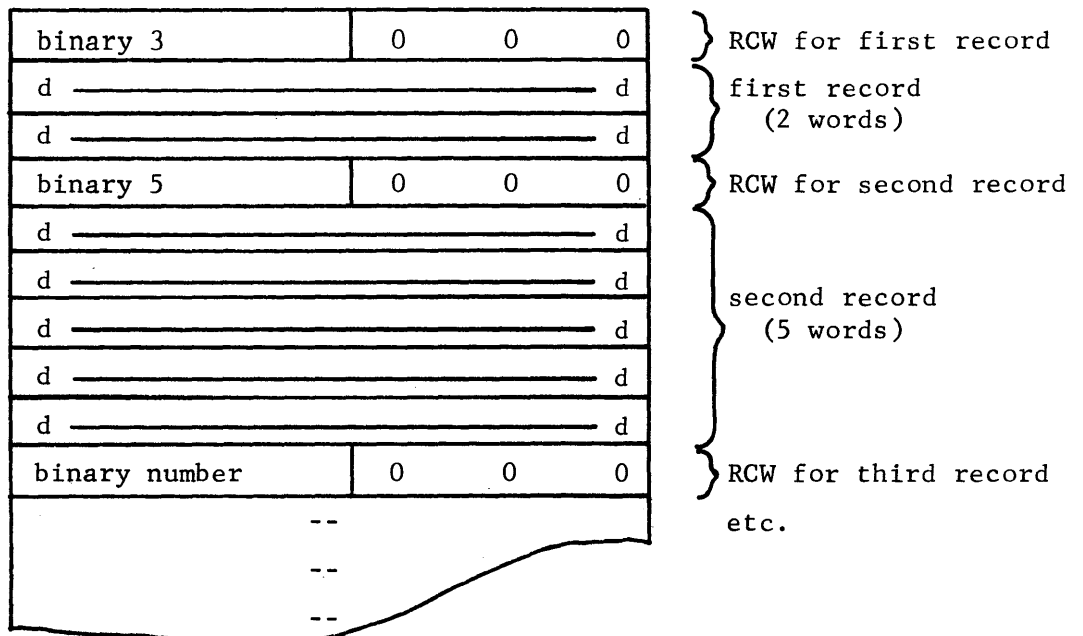
The code values may be any letter or digit, expressed as a non-numeric literal. The codes and mnemonic-names shown here are illustrative only.

2. In each RD entry in the Report Section, the optional CODE clause must be utilized when report codes are to be employed.

```
RD LEDGER WITH CODE CODE-1... .
  .
  .
  .
RD COST-DISTRIBUTION WITH CODE CODE-2... .
  .
  .
  .
etc.
```

The report codes enable the media conversion program to select from its many input records those which belong to the report currently being printed.

In a block of VLR records, only the record control word intervenes between the successive records. A VLR block beginning with a 12-character record followed by a 60-character record would appear as follows:



Recording mode must be binary for any file utilizing VLR format.

Block Size

GE-625/635 COBOL files using magnetic tape, disc, or drum are normally "blocked"; that is, each physical record on the peripheral medium is comprised of several logical records. Blocking enhances peripheral device performance in two ways: It saves peripheral operating time, and it saves space on the peripheral medium.

On magnetic tape, both savings result from fewer inter-record gaps. To process a given number of successive logical records, fewer starts and stops are needed, since a single physical read or write accounts for several logical records. Less time is spent traversing inter-record gaps. Since inter-record gaps are less frequent, they occupy a smaller percentage of the total length of the tape, so that a reel of tape can hold many more logical records. Very significant savings in both time and tape space result from blocking; when the file volume is large, savings are measured in numbers of reels.

On disc or drum, the time savings result primarily from fewer positioning actions on the rotating medium. The space savings result from the fact that actual physical record sizes on the magnetic media are fixed (at 320 words), so part of the device's storage capacity is wasted whenever a shorter physical record size is employed.

Block size is specified in COBOL via the BLOCK CONTAINS clause. Six possible cases must be considered:

<u>Optional use of BLOCK CONTAINS</u>	<u>FLR file</u>	<u>VLR file</u>
clause omitted	case 1	case 4
BLOCK CONTAINS...integer-3 RECORDS	case 2	case 5
BLOCK CONTAINS...integer-3 CHARACTERS	case 3	case 6

The following discussion presents GE-600 Series COBOL block size conventions, in terms of the number of computer words per block. The FLR formulas given here yield "net" block size, not reflecting block serial numbers. If block serial numbers are applied, the block size exceeds the "net" block size by one word; the user must allow for the extra word in planning his use of BLOCK CONTAINS.

FLR Files (cases 1-3)

1. BLOCK CONTAINS clause omitted--the overall block size will not exceed 320 words. Net block size is [the largest multiple of the record size not exceeding 320* words]. If a FLR file has 30-word records, net block size (with BLOCK CONTAINS omitted) would be 300 words.

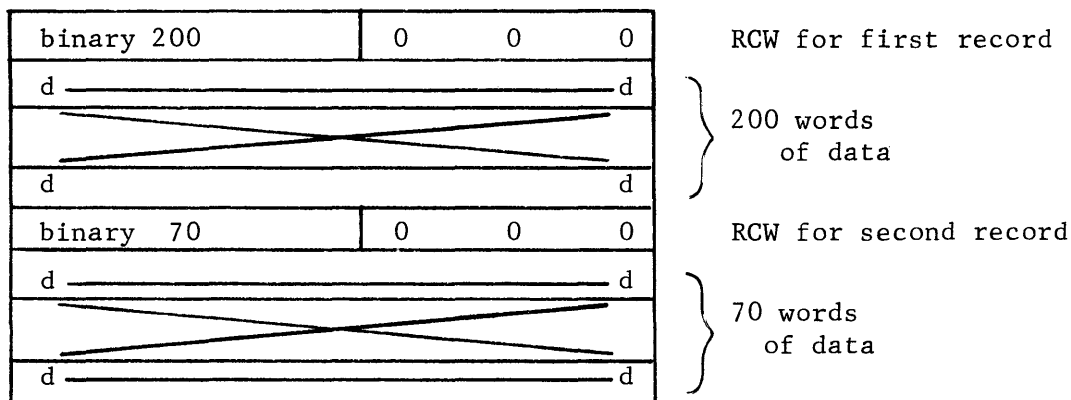
*319 words if block serial numbers are applied.

2. BLOCK CONTAINS...integer-3 RECORDS--the net block size is [integer-3 times the record size]. In the example given for case 1, the same result would be obtained by BLOCK CONTAINS 10 RECORDS.
3. BLOCK CONTAINS...integer-3 CHARACTERS--the block size will be as close to [integer-3 ÷ 6] words as possible. The net block size is [the largest multiple of the record size not exceeding (integer-3 ÷ 6)]. In the example given for case 1, the same result would be obtained by BLOCK CONTAINS 1800 CHARACTERS. Integer-3 should be exactly [6 times the overall record size in words times the desired blocking factor], plus 6 if block serial numbers are applied.

VLR files (cases 4-6)

4. BLOCK CONTAINS clause omitted--the maximum block size is 320 words (including block serial number, if applied). The actual block size will vary from one block to the next. In each output block, successive records are

space is insufficient to hold another record; the current block is then physically written out and a new block is begun. The physical block ends with the last word of the last record. If record sizes are 200 words and 70 words, respectively, a block might appear as follows:



The actual block size in this example is thus $[1 + 200] + [1 + 70] = 272$ words.

5. BLOCK CONTAINS...integer-3 RECORDS--the maximum net block size is $[(\text{maximum record size in words} + 1) \times \text{integer-3}]$. The increment of 1 to maximum record size allows for record control words. This convention allows at least integer-3 records per block; but the blocks are still built up as described for case 4. If not all records in a block are of the maximum size, more than integer-3 records may be included.

If the file has two record types, with sizes (in words) 5 and 75, respectively, and BLOCK CONTAINS 2 RECORDS, maximum block size is $2 \times [75 + 1] = 152$; the block might, however contain as many as 25 of the small records.

6. BLOCK CONTAINS...integer-3 CHARACTERS--the maximum block size is $[\text{the largest integer not exceeding } (\text{integer-3} \div 6)]$, including block serial number, if applied. Integer-3 should be a multiple of 6. Blocking conventions proceed as described for case 4.

The block size must be large enough to contain the file's largest logical record. When System Standard format is intended, the block size must not exceed 320 words, including the block serial number (see below). The compiler actually allocates sufficient buffer space to any System Standard format file to accommodate 320 word blocks. When System Standard format is not intended, however, magnetic tape files may have any desired block size not exceeding 4095 words.

Block Serial Number

The GE-600 Series System Standard format requires block serial numbers. In addition, block serial numbers may optionally be used on files which do not employ System Standard format, provided recording mode is binary.

When applied to a file, block serial numbers use the first computer word of each physical block of data. This word has the following format:

bit 0	17	18	bit 35
block serial number (binary integer)		block size in words (binary integer)	

The block serial number subfield contains the sequential number of this physical block within the current reel of this file (the current reel consideration pertains only to magnetic tape files).

The block size subfield contains the actual size of this block, excluding the block serial number control word itself.

Recording mode must be binary for any file to which block serial numbers are applied.

COBOL procedural statements cannot access the block serial number control word.

Specification and Handling of Labels

COBOL provides for the automatic handling of four types of labels - beginning and ending, file and tape. The notes for the LABEL RECORD clause in the File Description entry contain an indication of the relative position of these labels on the tape(s) associated with a file.

A label record is a logical record containing the labeling information about a tape or file. In order to have common recognition of these records, fixed names have been assigned to the four standard label record types:

BEGINNING-TAPE-LABEL
BEGINNING-FILE-LABEL
ENDING-FILE-LABEL
ENDING-TAPE-LABEL

The standard GE-625/635 Series BEGINNING-TAPE-LABEL and BEGINNING-FILE-LABEL have identical formats:

```
01 BEGINNING-TAPE-LABEL; SIZE 84 DISPLAY CHARACTERS.
02 LABEL-IDENTIFIER; PICTURE X(12) VALUE
   IS "GE 600 BTL".
02 INSTALLATION; PICTURE X(6).
02 REEL-SERIAL-NUMBER; PICTURE B9(5).
02 FILE-SERIAL-NUMBER; PICTURE B9(5).
02 REEL-NUMBER; PICTURE BB9999; RANGE
   IS "0001" THRU "9999".
02 DATE-WRITTEN; SIZE 6.
   03 LABEL-YEAR; PICTURE B99.
   03 LABEL-DAY; PICTURE 999; RANGE IS 001
     THRU 365.
02 FILLER; PICTURE BBB.
02 RETENTION-PERIOD; PICTURE 999;
   RANGE IS 001 THRU 999.
02 IDENTIFICATION; PICTURE X(12).
02 FILLER; SIZE 24 AN CHARACTERS.
66 ID RENAMES IDENTIFICATION.
```

The items within beginning labels have the following significance:

1. INSTALLATION is constant information for each user installation. This item is supplied automatically in output labels, but is ignored by input label checking routines.
2. REEL-SERIAL-NUMBER is the serial number of the physical tape reel. This number is also recorded externally on the reel itself. This item is supplied automatically in output labels.
3. FILE-SERIAL-NUMBER is the serial number of the first reel of the file. On the first reel, therefore, the values of FILE-SERIAL-NUMBER and REEL-SERIAL-NUMBER are identical. This item is supplied automatically in output labels. On input it is checked against an expected value unless an object-time option to bypass the check is exercised (see File Control Cards, below).
4. REEL-NUMBER is the number of the reel within the file. The first reel is number 0001, the second is 0002, etc. This item is automatically supplied in output labels and checked on input.

5. DATE-WRITTEN is the day-of-year on which the object-program has been executed to produce this file. This item is automatically supplied in output labels, but is ignored on input.
6. RETENTION-PERIOD is the number of days the file is to be retained. This item is ignored on input, but is processed in two distinct phases on each output file:
 - a. Every tape upon which a labeled output file is to be written is expected to have a prior label, which may be either a blank reel label or a beginning label on which the RETENTION-PERIOD has expired (current date minus DATE-WRITTEN exceeds RETENTION-PERIOD). These conditions are checked, and operator action is requested if they are not met. (See the GEFRC manual.)
 - b. If the user has specified a value for the RETENTION-PERIOD via the FD VALUE clause, this value is automatically supplied in the new output label, which replaces the prior beginning label on the output tape.
7. IDENTIFICATION is the literal name given to the file for external identification. On input, this item is automatically checked against the value specified via the FD VALUE clause; on output, the value specified in that clause is automatically supplied in the output label. If the VALUE OF IDENTIFICATION is not specified, these automatic procedures are bypassed.

The standard GE-600 Series ENDING-TAPE-LABEL and ENDING-FILE-LABEL also have identical formats:

```

01  ENDING-TAPE-LABEL; SIZE 84.
    02  SENTINEL; PICTURE X(6).
        88  END-OF-TAPE; VALUE IS " EOR  ".
        88  END-OF-FILE; VALUE IS " EOF  ".
    02  BLOCK-COUNT; PICTURE 9(6).
    02  FILLER; SIZE IS 72 AN CHARACTERS.

```

Except when RECORDING MODE IS BCD, BLOCK-COUNT has USAGE COMPUTATION. For BCD files, it is DISPLAY, as is implicitly shown above. BLOCK-COUNT is automatically supplied in output labels, and is checked on input labels against the computed block count.

The EOR value of SENTINEL causes an automatic tape swap on input, while the EOF value signals end of file.

Production of an ENDING-TAPE-LABEL on output is caused automatically when an end-of-tape foil is encountered. It may also be caused by a CLOSE REEL procedural statement. Production of this label is automatically followed by a tape swap. A CLOSE statement (without the REEL option) causes an ENDING-FILE-LABEL to be produced, along with any other CLOSE actions specified.

The FILLER items at the end of each of the above formats may be replaced by descriptions of additional data items to be included in the label records. Such additional items, if present, must be processed in USE procedures. Unless such items are included, the standard label descriptions are implicitly described by the LABEL RECORDS ARE STANDARD clause, and consequently standard labels need not be described, even if their contents are referenced in USE procedures.

When explicit label record descriptions are specified, the standard contents must be described exactly as shown above, with respect to data-names, PICTURES, and position. The overall size of each label record must be 84 characters. Departures from these rules can lead to unpredictable results. If label record descriptions are explicitly given for a file, their descriptions must precede the descriptions of logical data records. Standard label record names must never be mentioned in a DATA RECORD[S] clause.

VALUE clauses within label record descriptions do not result in automatic MOVES of the specified literals to output labels. All standard items within standard labels are automatically handled by the input/output routines; an option is provided for specifying the literals to be used with IDENTIFICATION and RETENTION-PERIOD via the FD VALUE clause.

If non-standard label formats are to be used in a file, then LABEL RECORDS ARE OMITTED must be specified, and the label records must be described as data records. Explicit procedures must then account for all label processing. Because of the complexities involved, use of non-standard records is impractical, and should be avoided if possible. (A compromise is sometimes possible--with USE BEFORE procedures the programmer may be able to transform non-standard input labels to standard format before the automatic checking takes place; and with USE AFTER procedures he may be able to transform standard output labels to non-standard format before they are written to the peripheral medium.)

A file with standard labels has the following overall format:

1. Beginning of medium indicator (Load Point reflective foil on magnetic tape)

2. BEGINNING-FILE-LABEL (or BEGINNING-TAPE-LABEL if this is not the first reel of the file)
3. End-of-file mark (octal 17 character)
4. Data blocks
5. End-of-file mark (octal 17 character)
6. ENDING-FILE-LABEL (or ENDING-TAPE-LABEL if this is not the last reel of the file)
7. End-of-file mark (octal 17 character)

On a multiple reel file, every reel except the last has an ENDING-TAPE-LABEL, while the last has an ENDING-FILE-LABEL. Similarly, every reel but the first has a BEGINNING-TAPE-LABEL, while the first has a BEGINNING-FILE-LABEL.

If LABEL RECORDS ARE OMITTED on an output file, each output tape is terminated with an end-of-file mark when the end-of-tape foil is detected, and a tape swap occurs.

If LABEL RECORDS ARE OMITTED on an input file, a special action occurs in GE-600 Series COBOL object-programs when an end-of-file mark is detected, since the standard means of recognizing logical end-of-file is not available. (Unless the file is a single-reel file, the user must determine via explicit procedures which reel terminates the file). When an end-of-file mark is detected, the appropriate AT END procedure is executed. If a subsequent READ occurs, a tape swap takes place and the first record of the next reel is obtained; otherwise, if the reel just ended is the last in the file, the file should be CLOSED.

Label record contents may be accessed only by USE procedures, and only one label record is available when USE procedures are executed. Standard label item data-names need never be qualified in procedural references.

Because label record formats are generally not the same for different computer lines, a program with explicit label record descriptions must usually be modified if it is to be compiled on any computer line other than that for which it was originally programmed. Not all fixed label item data-names can be used with all compilers, but the fixed label record names themselves are usually available, in addition to the following fixed data-names:

IDENTIFICATION
REEL-NUMBER
DATE-WRITTEN
BLOCK-COUNT
SENTINEL
END-OF-TAPE
END-OF-FILE

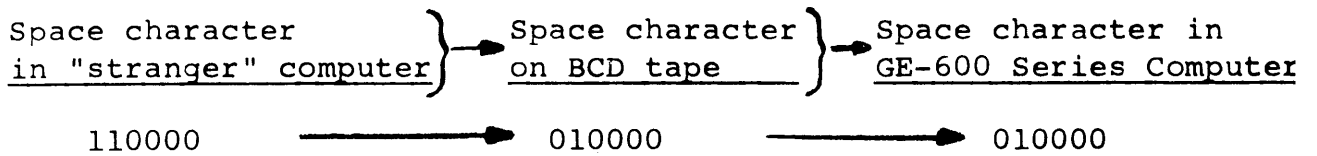
Although any file may be described as having standard labels, label records actually appear only on magnetic tape. The label checking and building provisions described above therefore actually occur only in processing magnetic tape files (including those intended for media conversion). Similarly, label-type USE procedures are executed only on magnetic tape files.

Recording Mode

The standard recording mode for GE-600 Series files is BINARY HIGH DENSITY. This mode is required for files having System Standard format. BINARY HIGH DENSITY is therefore the understood mode when the optional RECORDING MODE clause is omitted.

Internally in the GE-600 Series computer systems, all data values are represented by either binary numbers or by binary-coded internal "characters". For example, the letter A is represented internally by the six-bit binary code 010001. Use of the term "binary" in describing the recording mode reflects the fact that in that mode the peripheral medium represents the data with exactly the same binary bit configuration as in the computer memory. Any data which can be stored in memory can be written to and retrieved from a peripheral in the same configuration via binary mode. This is not true of BCD mode. BCD mode permits only binary coded characters, not binary numbers. The character with internal binary code 001010, whose graphic is "[" (left bracket), cannot be represented in BCD mode on magnetic tape.

GE-600 Series computer systems provide BCD mode for special compatibility purposes. Of the 64 possible binary-coded characters, a certain subset has wide usage for data processing applications in many machine lines. Included are the letters, the space character, the digits, and certain editing characters. These characters are represented by different binary codes on various machine lines, but their representation on BCD tape is standardized. For example, a space or blank has binary code 110000 in some machines, but has binary code 010000 in GE-600 Series computers. If a space character is written to magnetic tape in BCD mode on such a "stranger" machine, it will be read as a space character by the GE-600 Series computer.



The BCD and LOW DENSITY options need never be specified except for magnetic tape files, and are recommended only for compatibility purposes on magnetic tape. If the actual peripheral device in use is a printer or a card reader or punch, the input/output routines in the object-program automatically use the density appropriate to the device. If a file is intended for cards or printer via media conversion (either input or output), the recording mode utilized should be BINARY HIGH DENSITY.

Provided recording mode is BINARY, each record in the file may contain an arbitrary mixture of binary (COMPUTATIONAL or COMPUTATIONAL-n) and BCD or character-oriented information. Only BCD information may appear in BCD mode.

Many options in GE-600 Series COBOL require BINARY recording mode on the file affected:

1. System Standard format
2. VLR format
3. FOR CARDS or LISTING
4. Block serial numbers
5. REPORT[S]
6. OCCURS...DEPENDING
7. USAGE COMPUTATIONAL or COMPUTATIONAL-n
8. USAGE DISPLAY-2.

Multiple File Tapes

The MULTIPLE FILE option applies only to magnetic tapes. It permit two or more files to appear successively on the same physical reel of tape.

For COBOL object-programs, all files of a multiple file tape must actually be present. The tape is automatically positioned to the proper point each time a file is OPENed (as either INPUT or OUTPUT) Positioning is based on counting the number of logical files intervening between the desired file and the beginning of the tape, not on a label search. A file in POSITION 5 thus must be preceded on the tape by four prior files.

On an output tape, files must actually be written in the order in which they are to appear on the tape. For example, the POSITION 4

file cannot be written before the POSITION 2 file. However, the successive files may be produced by distinct object-programs. On an input tape, files may be processed in any order.

Two files on a multiple file tape cannot be OPEN concurrently. When an output file in a given POSITION has been OPENed, any files which might have previously existed in later POSITIONS are unavailable.

All of the files on a multiple file tape must have the same recording mode.

Either all of the files on a multiple file tape must be labeled, or else none may be. Labeled and un-labeled files cannot be mixed on such a tape.

When the files are labeled, the overall tape layout is as follows:

1. Beginning of tape indicator (Load Point reflective foil)
 2. BEGINNING-FILE-LABEL
 3. End-of-file mark
 4. Data blocks
 5. End-of-file mark
 6. ENDING-FILE-LABEL
 7. End-of-file mark
 8. BEGINNING-FILE-LABEL
 9. End-of-file mark
 10. Data blocks
 11. End-of-file mark
 12. ENDING-FILE-LABEL
 13. End-of-file mark
- etc.
- } first
(logical)
file
- } second
(logical)
file

When the files are not labeled, the overall tape layout is as follows:

1. Beginning of tape indicator
 2. Data blocks
 3. End-of-file mark
 4. Data blocks
 5. End-of-file mark
- } first file
- } second file
- etc.

COBOL rules require that all of the files on a multiple file tape must actually fit on one physical reel of tape.

The Peripheral Medium

GE-600 Series COBOL files can use drum, disc, magnetic tape, printer card reader, or card punch. (Files on drum and disc must be serially accessed via COBOL.)

Because their high speed is much better matched to the central processor's ability, the preferred media for object-program access are drum, disc, and magnetic tape. Although the printer, card reader, and card punch can be accessed directly by object-programs, it is preferable to employ media conversion, so that a file conceptually intended for cards or listing nevertheless uses a high speed peripheral when the object-program is executed. Standard media conversion programs provided as part of the GE-600 Series software system are small programs, requiring a minimum amount of memory and processor time; and they contain special provisions to obtain the highest possible operating speeds from the low speed peripherals.

In GE-600 Series COBOL, the source-program ASSIGNS each file to a symbolic file-code, not to a specific peripheral device. The actual device is specified separately via control cards when the object-program is scheduled for execution. A substantial degree of device-independence is possible; provided the System Standard format is used, a file can be assigned different storage media from one execution of the object-program to the next. The following exaggerated example illustrates the flexibility a daily report file can have:

1. On Monday, the file is assigned to an on-line printer.
2. On Tuesday, the file is assigned to mass storage, and a subsequent media conversion to printer is scheduled.

3. On Wednesday, the file is assigned to magnetic tape, and again a subsequent media conversion is scheduled.
4. On Thursday, the file is assigned to SYSOUT, so that subsequent media conversion is automatically scheduled.

If a file is intended for printer or a card device, the COBOL programmer can think of the file largely in terms of the intended device, despite the fact that a media conversion to or from tape or mass storage will probably be desired. He should specify FOR CARDS or LISTING in the SELECT sentence; and he should employ the System Standard format.

Certain data description options are not suitable for card or printer files; examples are COMPUTATIONAL (or COMPUTATIONAL-n) or DISPLAY-2 USAGES.

Use of standard label records is the preferred practice for all magnetic tape files, including those intended for cards or listing.

File Control Cards

Each COBOL file must be mentioned in a SELECT sentence in the FILE-CONTROL paragraph. In the SELECT sentence the file is assigned to a two-character symbolic file-code. When the object-program is scheduled for execution, each file-code must be associated with a peripheral device via a GECOS file control card; there will thus be one such card for each file selected except for optional files. (Sort-files are excluded from this rule.)

The file control cards are fully described in the GECOS manual. Those which apply to COBOL files are:

\$	DATA
\$	SYSOUT
\$	DISC
\$	DRUM
\$	PRINT
\$	TAPE
\$	READ
\$	PUNCH

When used with COBOL object-programs, each file control card must contain the file code to which the file is assigned. All cards other than \$ DATA and \$ SYSOUT require additional parameters to provide further information about the peripheral medium and the mode of processing.

Files intended for \$DATA and \$SYSOUT must have System Standard format. Other files entering or leaving the system via bulk media conversion normally utilize magnetic tape as the intermediate medium, and consequently require \$TAPE cards. Any file with block size exceeding 320 words must utilize magnetic tape.

Each file must of course be processed in a manner consistent with the peripheral device's nature. Those intended for \$SYSOUT, \$PRINT or \$PUNCH must be output; those intended for \$DATA or \$READ must be input. Provided \$TAPE or \$DISC or \$DRUM is used, a file may be processed as either input or output, and, after being closed, may be re-opened as either input or output, regardless how it was previously processed. This capability is particularly useful when a "scratch" file is needed within a program--the file may be processed as output, then reOPENed and processed as input within the same program.

When a \$TAPE control card is used, it must contain the FILE-SERIAL-NUMBER (see Specification and Handling of Labels, above). Sometime the value of this label item cannot be known in advance. In such a case, the special value 99999 can be punched; this value causes the automatic FILE-SERIAL-NUMBER check to be bypassed.

System Standard Format

To permit a substantial degree of peripheral device independence, a System Standard format has been defined for GE-600 Series data files. A file with the System Standard format has the following properties:

1. Data blocks may vary in length up to but not over 320 words, including block serial numbers.
2. Block serial numbers are applied.
3. Recording mode is binary high density.
4. VLR format is applied.
5. Label records are standard.

For convenience, GE-600 Series COBOL permits the user to specify APPLY SYSTEM STANDARD FORMAT in the I-O-CONTROL paragraph. When this APPLY clause is used, the BLOCK CONTAINS and RECORDING MODE clauses may be omitted, and it is unnecessary to APPLY VLR or BLOCK SERIAL. The LABEL RECORDS clause, however, must appear in every FD entry in the FILE SECTION, according to COBOL rules.

As an alternative to specifying APPLY SYSTEM STANDARD, the user may obtain the System Standard format by an appropriate combination of other descriptive clauses. He would still omit BLOCK CONTAINS and RECORDING MODE, but must APPLY BLOCK SERIAL. If the file is such that FLR format would normally be applied automatically, System Standard format nevertheless requires VLR, which may be specified explicitly via APPLY VLR (see Logical Record Format, which appears earlier in this paper).

Magnetic tape files may depart in any respect from the System Standard format. For example, a large master file which is definitely planned for magnetic tape might employ block sizes larger than 320 words. A block size 3 times this large (960 words), for example, would increase a tape's capacity by about 20%.

System Standard format must be used when device independence is intended. System Standard format must be used for all serially accessed files in mass storage. Particularly important for serial files in mass storage are the 320 word block limit, block serial numbers, and VLR format. (Binary high density is the only applicable recording mode, and label records--although conceptually standard--do not actually appear in mass storage.) System Standard format must be used for files intended for SYSOUT.

An additional significance of System Standard format is that the "standard" properties are generally assumed unless contrary options are specified. In most GE-625/635 Series software, the user may omit the relevant file parameters in the description of a System Standard file. In GE-625/635 Series COBOL, for example, the standard block size and recording mode are assumed when the BLOCK CONTAINS and RECORDING MODE clauses are omitted. System Standard format thus provides a convenient shorthand method for describing GE-625/635 Series files.

Summary of File Property Relationships

The following charts summarize the relationships of the various options for COBOL files.

specified property	conflicts with																					
	FOR CARDS	FOR LISTING	BLOCK SERIAL	SYSTEM STANDARD	VLR	MULTIPLE FILE	RECORDING MODE omitted	BINARY HIGH DENSITY	BINARY LOW DENSITY	BCD (either density)	BLOCK clause omitted	BLOCK...RECORDS	BLOCK...CHARACTERS	LABEL...STANDARD	LABEL...OMITTED	VALUE OF label item	single DATA RECORD	several DATA RECORDS, sizes*	REPORT[S]	REPORT[S] and DATA RECORD [S]	USAGE not DISPLAY [-1]	OCCURS...DEPENDING
FOR CARDS		X								X									X	X	X	X
FOR LISTING	X									X											X	X
BLOCK SERIAL										X												
SYSTEM STANDARD									X	X				X								
VLR										X												
MULTIPLE FILE																						
RECORDING MODE omitted										X	X											
BINARY HIGH DENSITY										X	X											
BINARY LOW DENSITY				X		X	X			X												
BCD (either density)	X	X	X	X	X	X	X	X										X	X	X	X	X
BLOCK clause omitted												X	X									
BLOCK...RECORDS											X	X						*	*	*	*	
BLOCK...CHARACTERS										X	X											
LABEL...STANDARD														X								
LABEL...OMITTED				X										X	X							
VALUE OF label item														X								
single DATA RECORD																	X					
several DATA RECORDS, sizes*										X	*					X						
REPORT [S]	X									X	*										X	
REPORT [S] and DATA RECORD [S]	X									X	*										X	
USAGE not DISPLAY [-1]	X	X								X									X	X		
OCCURS...DEPENDING	X	X								X	*											

* Use of the CHARACTERS option is preferred when record sizes are not uniform.

property

implies

	BLOCK SERIAL	VLR	FLR	BINARY HIGH DENSITY	System Standard	Block Size (≥ 320 words)	Card Medium Code	Printer Medium Code	PROCESS AREA
APPLY PROCESS AREA									X
FOR CARDS		X			X		X		X
FOR LISTING		X			X			X	X
BLOCK SERIAL	X								
SYSTEM STANDARD	X	X							
VLR		X							
MULTIPLE FILE									
RECORDING MODE omitted				X					
BINARY HIGH DENSITY				X					
BINARY LOW DENSITY									
BCD (either density)									
BLOCK clause omitted						X			
BLOCK...RECORDS									
BLOCK...CHARACTERS									
LABEL...STANDARD									
LABEL...OMITTED									
VALUE OF label item									
SAME RECORD AREA									X
single DATA RECORD									
several DATA RECORDS, sizes \neq									
REPORT[S]		X						X	
REPORT[S] and DATA RECORD[S]		X						X	X
USAGE not DISPLAY[-1]									
OCCURS...DEPENDING									X
USAGE COMPUTATIONAL[-n] requiring double-word precision									X

* Implied in the absence of other overriding properties.

property	requires						
	BLOCK SERIAL	FLR	VLR	BINARY mode	HIGH DENSITY	LABEL...STANDARD	BLOCK size \leq 320 words
FOR CARDS			X	X			
FOR LISTING			X	X			
BLOCK SERIAL				X			
SYSTEM STANDARD	X			X	X	X	X
VLR				X			
MULTIPLE FILE							
RECORDING MODE omitted							
BINARY HIGH DENSITY							
BINARY LOW DENSITY							
BCD (either density)		X					
BLOCK clause omitted							
BLOCK...RECORDS							
BLOCK...CHARACTERS							
LABEL...STANDARD							
LABEL...OMITTED							
VALUE OF label item						X	
single DATA RECORD							
several DATA RECORDS, sizes†			X	X			
REPORT [S]			X	X			
REPORT [S] and DATA RECORD [S]			X	X			
USAGE not DISPLAY [-1				X			
OCCURS...DEPENDING			X	X			

INTERNAL FORMAT CONSIDERATIONS

Methods of internal data representation differ considerably between various computer lines. A given application, however, can usually be accomplished by the machines in most computer lines. COBOL data descriptions are therefore expressed in terms of a "standard data format", which is oriented to the problem or application rather than to a particular machine line.

Each item is described as if it consists of a string of contiguous data characters. Each character, whether it is a letter, a decimal digit, or a punctuation or editing character, conceptually occupies a single position. The "size" of an item is the number of conceptual characters comprising the item. Within any succession of related items, each conceptually follows immediately after its predecessor. (A logical record is the largest consecutive set of related information.)

The standard data format is closely related to the internal data representation employed by GE-600 Series COBOL object-programs. Within the computer system, data storage is organized into 36-bit words. Each word is subdivided into six 6-bit character positions. Every character position can store the binary-coded representation of one data character.

When USAGE is DISPLAY or DISPLAY-1, an elementary item's conceptual size (in standard data format characters) is the same as the number of character positions the item occupies in storage.

The 6-bit binary coding scheme allows 64 combinations of bit values, so that 64 characters may be represented by unique binary codes. A specific graphic symbol has been assigned to each of the 64 unique binary codes, yielding the standard GE-600 Series character set. When the problem requires the conceptual value of a character to be, say, dollar sign (\$), the actual representation of that character within the computer system is the standard binary code for dollar sign, which is 101011. The relationship of standard graphic symbols with conceptual character values is made concrete by the printer, which prints the standard graphic for each binary coded internal character. (Certain codes reserved for printer control are excluded.)

For DISPLAY and DISPLAY-1 items, the standard binary code is used within the computer system to represent each data character, except in the case of SIGNED NUMERIC items (including any DISPLAY items whose PICTUREs contain the S symbol). If the value of such an item is non-negative, the data characters employ the standard binary codes. If the value is negative, the sign is expressed by a special variation

in the binary code of the least significant digit; specifically, all bits except the 2^5 bit have the usual values, but the 2^5 bit is set to 1. (This convention corresponds to the punched card convention of no overpunch on non-negative values, with an "eleven" overpunch over the low-order digit of a negative value.)

The only respect in which the internal representation of DISPLAY-2 items differs from that of DISPLAY items is that a special 6-bit binary code is used for each DISPLAY-2 character, not the standard code.

Three special COBOL features cause departures from the normal close relationship of standard data format and actual internal data representation:

1. SYNCHRONIZED, which can only be specified for an elementary item, usually results in fill characters preceding and/or following the item. Thus the internal formats of groups containing a SYNCHRONIZED item are affected, although the item itself is not affected. SYNCHRONIZED is the means in COBOL for relating elementary items to computer words, to enhance processing efficiency.
2. COMPUTATIONAL (or COMPUTATIONAL-n) causes any item with such a USAGE to be represented as a 1-word or 2-word binary number. A COMPUTATIONAL[-n] item is oriented to computer words, so that fill characters will intervene unless the item just preceding a COMPUTATIONAL[-n] item terminates in the last character position of the word. Furthermore, a two-word ("double-precision") COMPUTATIONAL[-n] item must begin an even number of words from the first word of the record which contains it, so that an additional full word of fill characters may sometimes be supplied by the compiler to meet this requirement.
3. OCCURS...DEPENDING causes a special format to be used for storing affected files on peripheral media.

The internal format of each overall data record may differ from the conceptual format in certain respects. Each record actually occupies an integral number of computer words, so fill characters may be required in the last position of the last word. Furthermore, there may be other character fill within the internal format of a record as well as word filler if required by "2." above.

A given record description results in the same internal format, whether the record appears in the File, Working-Storage, or Constant Section.

External Versus Internal Formats

In COBOL, the data representation within the computer memory is called the "internal format", while the data representation on peripheral media is called the "external format".

A special external format is used for records in a file when OCCURS...DEPENDENT has been specified in the record descriptions. Otherwise, the external format on disc, drum, or a magnetic tape recorded in binary mode is exactly the same as the internal format.

On BCD tape, the formats are the same except that the 6-bit binary codes representing the data characters are generally different from the codes used in memory.

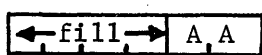
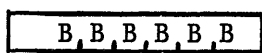
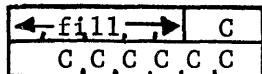
On printer listings, the conceptual formats are of course realized in the printed graphics, and external and internal formats are equivalent except for two differences--each line must terminate with non-printing slew-control characters, and horizontal tabulations of up to 120 character positions can be summarized in a single pair of characters in memory.

External and internal formats are equivalent for punched cards.

Result of SYNCHRONIZED Clause

A SYNCHRONIZED item monopolizes the smallest number of consecutive whole words that can contain it.

SYNCHRONIZED RIGHT causes the item to be oriented in such a way that its last character occupies the last position of the final word. Unless the item's size is a multiple of six characters, fill characters precede the first data character of the item in the unused positions of the first word, as the following examples illustrate:

Item's Description	Resulting Orientation	
03 A; SIZE 2; SYNCHRONIZED RIGHT.		(Internal And External)
03 B; SIZE 6; SYNCHRONIZED RIGHT.		
03 C; SIZE 7; SYNCHRONIZED RIGHT.		

SYNCHRONIZED LEFT causes the item to be oriented in such a way that its first character occupies the first position of the initial word. Unless the item's size is a multiple of six characters, fill characters follow the last data character of the item in the unused positions of the last word, as the following examples illustrate:

Item's Description	Resulting Orientation														
03 D; SIZE 2; SYNCHRONIZED LEFT.	<table border="1"><tr><td>D</td><td>D</td><td>←fill→</td></tr></table>	D	D	←fill→											
D	D	←fill→													
03 E; SIZE 6; SYNCHRONIZED LEFT.	<table border="1"><tr><td>E</td><td>E</td><td>E</td><td>E</td><td>E</td><td>E</td></tr></table> (Internal And External)	E	E	E	E	E	E								
E	E	E	E	E	E										
03 F; SIZE 7; SYNCHRONIZED LEFT.	<table border="1"><tr><td>F</td><td>F</td><td>F</td><td>F</td><td>F</td><td>F</td><td>F</td></tr><tr><td>F</td><td>←fill→</td><td></td><td></td><td></td><td></td><td></td></tr></table>	F	F	F	F	F	F	F	F	←fill→					
F	F	F	F	F	F	F									
F	←fill→														

Because SYNCHRONIZED items never share the words they occupy with other items, fill characters may be required in the word preceding the first word of a SYNCHRONIZED item, as the following example illustrates:

Item's Description	Resulting Orientation																														
01 G.	<table border="1"> <tr><td>H</td><td>H</td><td>H</td><td>H</td><td>H</td><td>H</td></tr> <tr><td>H</td><td>H</td><td>I</td><td>←fill→</td><td></td><td></td></tr> <tr><td>←fill→</td><td>J</td><td>J</td><td>J</td><td></td><td></td></tr> <tr><td>K</td><td>K</td><td>K</td><td>K</td><td>L</td><td>←fill→</td></tr> <tr><td>M</td><td>M</td><td>M</td><td>M</td><td>M</td><td>M</td></tr> </table> (Internal And External)	H	H	H	H	H	H	H	H	I	←fill→			←fill→	J	J	J			K	K	K	K	L	←fill→	M	M	M	M	M	M
H		H	H	H	H	H																									
H		H	I	←fill→																											
←fill→		J	J	J																											
K		K	K	K	L	←fill→																									
M		M	M	M	M	M																									
02 H; SIZE 8.																															
02 I; SIZE 1.																															
02 J; SIZE 3; SYNCHRONIZED RIGHT.																															
02 K; SIZE 4.																															
02 L; SIZE 1.																															
02 M; SIZE 6; SYNCHRONIZED LEFT.																															

It is very important to understand that the fill characters just discussed are implied by the combination of the item's size and the SYNCHRONIZED clause. The fill character positions cannot be described (except in a roundabout way via REDEFINES). An attempt to specify such fill characters via FILLER will lead to undesired consequences, as the following example illustrates:

Item's Description	Resulting Orientation																								
01 P.	<table border="1"> <tr><td>Q</td><td>Q</td><td>Q</td><td>Q</td><td>Q</td><td>Q</td></tr> <tr><td>Q</td><td>Q</td><td>R</td><td>F</td><td>I</td><td>L</td></tr> <tr><td>L</td><td>E</td><td>R</td><td>←fill→</td><td></td><td></td></tr> <tr><td>←fill→</td><td>S</td><td>S</td><td>S</td><td></td><td></td></tr> </table> (Internal And External)	Q	Q	Q	Q	Q	Q	Q	Q	R	F	I	L	L	E	R	←fill→			←fill→	S	S	S		
Q		Q	Q	Q	Q	Q																			
Q		Q	R	F	I	L																			
L		E	R	←fill→																					
←fill→		S	S	S																					
02 Q; SIZE 8.																									
02 R; SIZE 1.																									
02 FILLER; SIZE 6.																									
02 S; SIZE 3; SYNCHRONIZED RIGHT.																									

To minimize the amount of leading fill resulting from SYNCHRONIZED items, grouping such items together within a record description is recommended (provided circumstances permit such an arrangement), rather than scattering SYNCHRONIZED items about among non-SYNCHRONIZED items.

The fill characters resulting from SYNCHRONIZED appear in both the internal and external record formats.

Result of COMPUTATIONAL Usages

A special type of word-synchronization is applied to COMPUTATIONAL[-n] items. As for DISPLAY[-n] items which are SYNCHRONIZED, each COMPUTATIONAL[-n] item monopolizes the computer word or words which contain it, so that fill characters may be required to complete the word preceding a COMPUTATIONAL[-n] item. Unlike DISPLAY[-n] items, however, COMPUTATIONAL[-n] items fully occupy the word or words which contain them.

COMPUTATIONAL[-n] items occupying one word are:

1. COMPUTATIONAL items in one of the following categories:


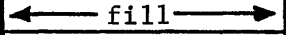
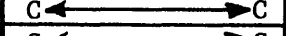
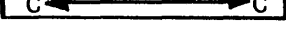
Number of Integral Digits	Number Of Fractional Digits
1-8	0
0-5	1-3
0-3	4-5
0	6-8

2. COMPUTATIONAL-1 or COMPUTATIONAL-2 items with no more than 8 digits specified in their descriptions.
3. COMPUTATIONAL-3 items.

COMPUTATIONAL[-n] items occupying two words are:

1. COMPUTATIONAL items not in one of the above categories.
2. COMPUTATIONAL-1 or COMPUTATIONAL-2 items with more than 8 digits specified in their description.

A one-word COMPUTATIONAL[-n] item may occupy any word of a record. A two-word COMPUTATIONAL[-n] item, however, must begin an even number of words from the beginning of the record which contains it. A word of fill may be supplied for such a two-word item to enforce this convention, as the following example illustrates.

Item's Description		Resulting Orientation
01	A.	
02	B; PICTURE 99V99; COMPUTATIONAL.	 (Internal
02	C; PICTURE 9(10)V99; COMPUTATIONAL.	 And
		 External)

Because of the convention just described, it is recommended that two-word COMPUTATIONAL[-n] items be grouped together within a record description when circumstances permit, and that all COMPUTATIONAL[-n] items be grouped with SYNCHRONIZED items.

COMPUTATIONAL[-n] items are not stored in a manner related to the character position subdivisions of a computer word. Instead, such items are stored as follows:

1. COMPUTATIONAL and COMPUTATIONAL-2 items utilize the GE-600 Series binary floating point format.
2. COMPUTATIONAL-1 items are stored as one-word or two-word binary integers.
3. COMPUTATIONAL-3 items are stored as one-word fixed point binary integer numbers.

Since a COMPUTATIONAL[-n] item as stored has no character orientation, it does not make sense to attempt to manipulate it as if it were comprised of characters. Thus, the storage area may be redefined for some distinct purpose, but not, for example, to give separate access to the integral and fractional parts of a COMPUTATIONAL item. For similar reasons, a group MOVE involving COMPUTATIONAL[-n] items should normally entail only sending and receiving groups with similar descriptions; MOVE CORRESPONDING may be used otherwise. COBOL rules do not require adherence to the suggestions given in this paragraph, but it is the user's responsibility to assure that his application of a group MOVE or REDEFINE makes sense. (The compiler produces warning comments to draw attention to certain types of suspicious cases.)

Result Of OCCURS...DEPENDING

In GE-600 Series COBOL, a special external format is utilized for

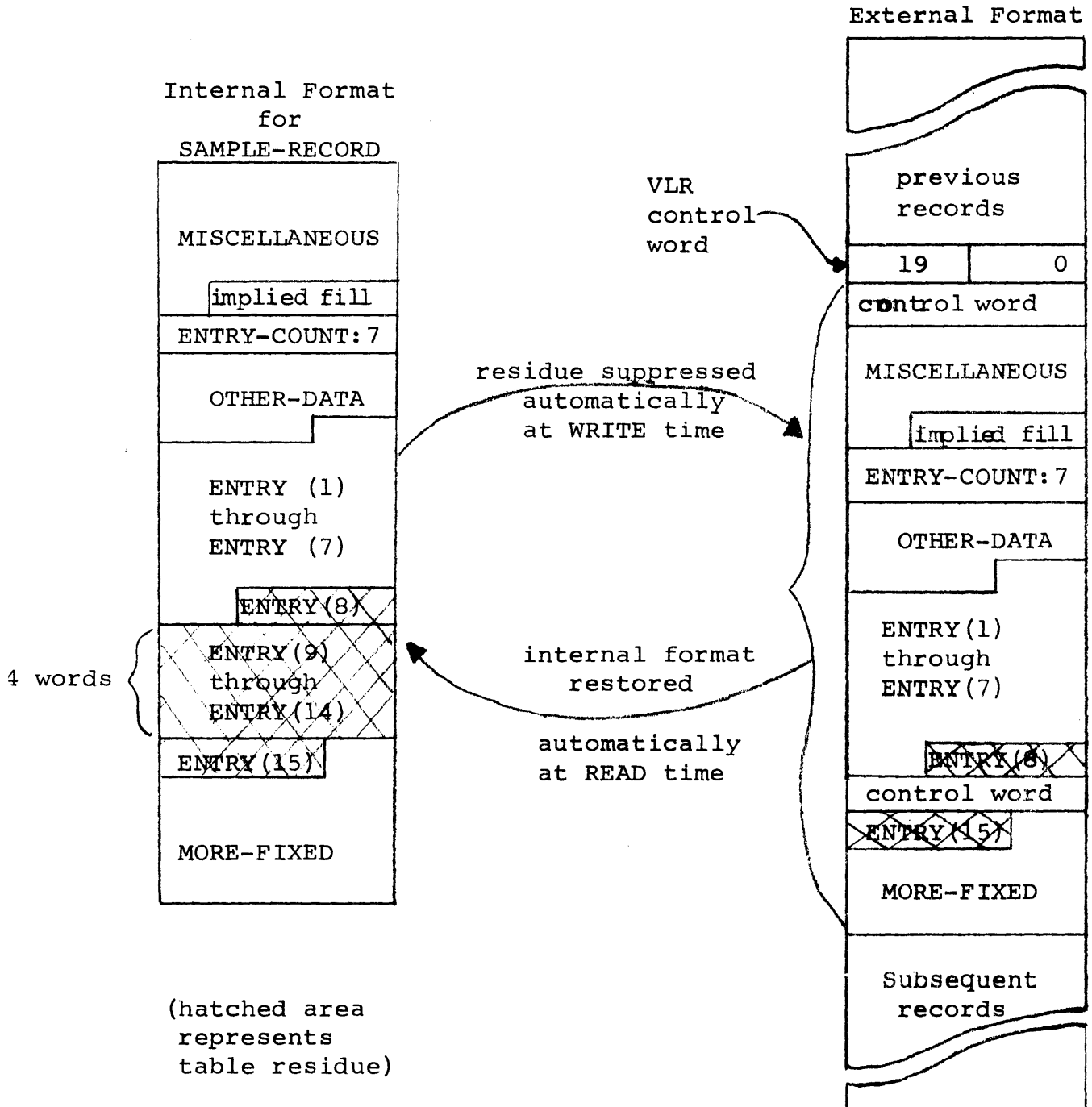
files in which the OCCURS...DEPENDING option is used in the record description entries. Each table described by OCCURS...DEPENDING is considered to vary in length (that is, in the number of consecutive significant items the table contains). In its internal format, such a table is allocated a storage area of size sufficient to hold the maximum number of occurrences. The storage area intervening between the last significant table item and the end of the table is called the "table residue". A variable-length table thus presents an opportunity for suppression of the table residue in an output record at WRITE time; and the record must be returned to the standard internal format at READ time.

If OCCURS...DEPENDING appears in any record description of a file, the compiler automatically applies a PROCESS AREA to the file. Each record appears in the PROCESS AREA in the standard internal format whenever the object-program has access to it. WRITE causes each suppression opportunity to be evaluated; if two or more whole words can be suppressed, the opportunity is accepted, while otherwise it is refused. Residue suppression thus proceeds on a multiple-whole-word basis. Each residue area suppressed is replaced by a control word in the external format. In addition, every record in such a file, whether or not it contains a variable-length table, begins with a special control word in its external format.

Consider a data record described as follows:

```
01  SAMPLE-RECORD.
    02  FIXED-PORTION.
        03  MISCELLANEOUS; PICTURE X(20).
        03  ENTRY-COUNT; SIZE 2 COMPUTATIONAL-1.
        03  OTHER-DATA; PICTURE X(16).
    02  VARIABLE-LENGTH-TABLE.
        03  ENTRY; PICTURE X(4); OCCURS 1 TO
            15 TIMES DEPENDING ON
            ENTRY COUNT.
    02  MORE-FIXED; PICTURE X(20).
```

The following illustration shows the relationship of the internal and external formats of SAMPLE-RECORD when the value of ENTRY-COUNT happens to be seven:



FILE STORAGE AREAS

In the object-program, a "buffer area" must be available for each file. The buffer area size exceeds the maximum overall block size by one word. Two buffers are automatically assigned to each file selected in the source-program.

Just one buffer area may be allocated to any file via the RESERVE NO ALTERNATE AREA option in the SELECT sentence. If only a single buffer is employed, input or output operations for the file cannot proceed concurrently with execution of the object-program. During input/output operations on such a file, control is therefore passed to the operating system, to allow other active object-programs to utilize the central processor. If an alternate buffer is reserved, however, the object-program can continue processing data in the current buffer concurrently with the execution of input/output operations involving the alternate buffer. Alternating buffers can therefore contribute to object-program efficiency, particularly on large-volume files. Decisions as to which files to buffer and which not should be based upon such considerations as file volume and effective use of memory within a multi-programming environment.

GE-600 Series COBOL provides a special optional file processing method in the form of the PROCESS AREA option. A PROCESS AREA is a memory area outside the buffers, large enough to hold the file's largest logical record. When a file using a PROCESS AREA is input, each logical record is implicitly moved to the PROCESS AREA from the input buffer when the READ statement is executed, and all other procedures reference the record in the PROCESS AREA. When such a file is output, each logical record is implicitly moved from the PROCESS AREA to the output buffer when the WRITE statement is executed, and all other procedures similarly reference the record in the PROCESS AREA.

A PROCESS AREA may be applied whether one or two buffer areas are allocated. Several source-language options cause the compiler to apply a PROCESS AREA:

1. APPLY PROCESS AREA
2. FOR LISTING/FOR CARDS
3. OCCURS...DEPENDING
4. SAME RECORD AREA (see below)
5. Both REPORT[S] and DATA RECORD[S] (the data records utilize the PROCESS AREA).
6. Double precision COMPUTATIONAL[-n] items in record
7. Sort files

When no PROCESS AREA is applied (implicitly or explicitly), the current data record is processed right in the buffer. Since record origins in the buffer generally differ from one record to the next, the record contents must be addressed relatively on the basis of the current record origin. Extra housekeeping is involved in processing such a record, because of the relative addressing required. With a PROCESS AREA applied, all record contents will occur in fixed positions, so that relative addressing and consequent extra overhead are not involved. For a file with heavy processing activity, savings in both time and space can result from applying a PROCESS AREA. It should be noted, however, that WRITE FROM and READ INTO statements are not efficient when applied with a PROCESS AREA.

PROCESS AREA also leads to more effective blocking on VLR output files. When no PROCESS AREA is applied to such a file, each output block is physically written as soon as the remaining buffer space is insufficient to hold the largest record of the file; this means that actual block sizes can run substantially smaller than the file's maximum block size. When PROCESS AREA is applied, however, the output block is physically written only when the remaining area is too small for the current logical record. With PROCESS AREA, therefore, one or more extra records can often be fit into an output block.

A buffer area, or a pair of alternating buffer areas, can be shared by two or more files via the SAME AREA clause. When this clause has been specified, the compiler evaluates the maximum buffer requirement of all of the files, and allocates adequate buffers to the first file to handle any of the others. If only one buffer is to be used for each file, a single buffer will be allocated; if two buffers are to be used for any of the files, two buffers will be allocated, but the single- or double-buffering for each file is still determined by the individual file's RESERVE option.

SAME AREA is not a recommended method for avoiding moves on a master file being updated by the object-program. Indeed, the result of using SAME AREA on files which are to be OPEN concurrently may be unpredictable unless procedures are carefully planned to avoid conflicts.

A PROCESS AREA may be shared by two or more files via the SAME RECORD AREA option. When this option is specified, the compiler allocates to the first file a PROCESS AREA adequate in size for the largest logical record of all of the files involved, and causes all of the files to share this PROCESS AREA. (The PROCESS AREA is applied in this case whether or not it has been explicitly specified.) SAME RECORD AREA does not result in buffer sharing. SAME RECORD AREA is recommended for minimizing moves on a master file being updated, particularly if OCCURS...DEPENDING is also used. When SAME RECORD

AREA is used for this purpose, procedures must make special arrangements to avoid deleting input records when insertions are required.

A file may have one or two buffer areas in any object-program, regardless how many it has in other object-programs. Similarly, a file may participate in SAME AREA or SAME RECORD AREA sharing in an object-program without regard to whether or not it is involved in these options in other object-programs. Except when the file description implies a PROCESS AREA, this option may also be exercised or omitted independently in each object-program.

Provided the rules for segmentation are followed, the compiler arranges communications so that a single set of storage areas is allocated to each file in a segmented program, and each subprogram references the same storage area at execution time.

APPENDIX C
PROCESSING NONLABELED MULTIPLE REEL FILES

According to the official COBOL rules, a file must not be READ after an AT END return unless it has been CLOSED and reOPENed. It is not possible for the compiler itself to detect violations of this rule, so no such check is attempted. In fact, a special GEFRC feature makes use of READ after AT END quite important in certain circumstances.

Consider a multiple reel file on which LABEL RECORDS ARE OMITTED. When the object-program is executed, GEFRC will have no way to tell from the tape contents which reel is the last. However, the user can easily provide this function via COBOL procedures. He must first arrange a special means of finding out how many reels to expect; a suitable approach would be to ACCEPT from GEIN a parameter giving the reel-count. The following example then illustrates how to achieve proper tape swapping and "true" end-of-file detection:

```
      .  
      .  
      .  
A.   READ file-name AT END GO TO EOF-CHECK.  
      .  
      .  
      .  
  
      EOF-CHECK.  SUBTRACT 1 FROM REEL-COUNT;  IF REEL-COUNT IS NOT  
                  ZERO GO TO A; ELSE CLOSE file-name  THEN GO TO ENDING-  
                  ROUTINE.
```

When GEFRC reaches the physical end-of-file mark at the end of each reel of the file it will take the AT END return for the active READ. Execution of another READ then causes a tape swap, and obtains the first data record from the next reel. Execution of the CLOSE, on the other hand, entails standard nonlabeled file closeout conventions.

A nonlabeled tape is often terminated simply with a physical end-of-file mark. For a multiple reel file which is so constructed, the technique outlined above is the only practical way to accomplish tape swapping. If, however, each reel concluded with a special data record whose value signalled end-of-reel, tape swapping could be accomplished via the COBOL CLOSE REEL feature, used in connection with suitable IF tests.

The user should understand that tape swapping via READ after AT END is a special GE-625/635 Series feature provided by GEFRC, and may not be available on other computer lines.

APPENDIX D
PROCESSING STRANGER FILES VIA COBOL

INTRODUCTION

GE-600-Line software provides the general capability to process stranger files in COBOL object-programs. A stranger file is one which for any reason cannot be fully described to COBOL; possible reasons are:

1. The character set of the data as stored in the peripheral medium does not coincide with character sets known to GE-625/635 COBOL.
2. Data records are not arranged to occupy multiples of six characters.
3. Binary fixed- or floating-point numbers were developed on a computer system using signed-magnitude arithmetic or different exponent sizes from GE-625/635 requirements.
4. Variable length records do not have record control words in the format required by GEFRC; e.g., variable length records appear on a BCD-mode tape.
5. Bit-coded parameters exist in the file.

If a file has any of the first four properties listed above, it is undoubtedly produced by or for a computer line other than the GE-625/635 computers. Files with any of these properties, or many others, are accessible to COBOL object-programs via the techniques described in this appendix.

Objectives

For the purposes of this appendix, it is assumed that "pure" COBOL is to be emphasized; the ENTER verb is not to be the answer to stranger file processing.

The techniques described must permit the COBOL-oriented applied programmer to proceed as if the stranger file were an ordinary GE-625/635 COBOL data file.

Furthermore, the techniques must not entail user modifications to the COBOL compiler or to GEFRC.

When stranger hardware, such as an off-line printer or a satellite computer is to be an integrated part of an installation's machine complement, the stranger file techniques must permit all object-programs to utilize the stranger hardware automatically, without human concern or intervention on a program-by-program basis.

TARGET FORMATS

The objectives imply that data formats seen by any COBOL object-program must be standard GE-625/635 COBOL data formats. Format translation must therefore occur between COBOL object-programs and hardware reads and writes. The following requirements must be satisfied:

1. On an input file, the COBOL object-program will request one record at a time, using .GAGET and the standard calling sequences. When control is returned to the object-program, the record must be where the object-program expects to find it, in the format expected by the object-program, regardless how the record appeared in the stranger file.
2. On an output file, the COBOL object-program will present one record at a time for output, using .GAPUT or .GAWTR (and, if necessary, .GAPTS) with a standard calling sequence. The record is in the GE-625/635 COBOL format understood by the object-program and in a standard place, regardless of how it is to appear in the stranger output file. When control is returned to the object-program, the record must have been so processed that the object-program is no longer concerned with it.

With these requirements understood, techniques for stranger file processing via COBOL can be discussed.

FORMAT TRANSLATION PROVISIONS

Alternate Entry Symbols for GEFRC Modules

GEFRC provides dual global symbols (SYMDEF's) for many key entry points:

<u>Standard Symbol</u>	<u>Alternate Symbol</u>
OPEN	.GAOPE
CLOSE	.GACLS
GET	.GAGET
PUT	.GAPUT
PUTSZ	.GAPTS
WTREC	.GAWTR
IOEDIT	.GAEDI
PRINT	.GAPRN
PUNCH	.GAPNC

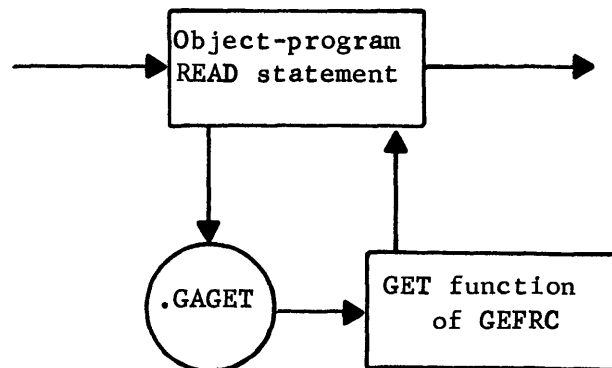
The two symbols for each entry point are entirely equivalent. By convention, however, COBOL object-programs will utilize the alternate symbols (prefix ".G") rather than the standard symbols.

Plug-in-Points

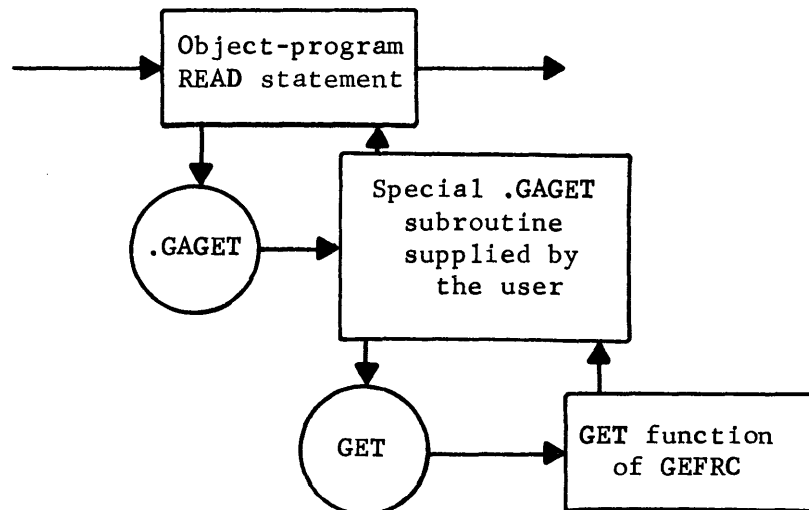
Each alternate symbol utilized by a COBOL object-program defines a point at which a special subroutine may be inserted or "plugged in" between the COBOL object-program and the GEFRC module which the object-program "thinks" it is calling. This plug-in is accomplished by using the appropriate GEFRC alternate symbol as the entry symbol or the special subroutine. When such a subroutine is present, it automatically intercepts all object-program calls to the GEFRC entry point in question.

For example, consider the GET function of GEFRC, used for COBOL READ statements. The object-time execution sequence may be readily diagrammed for various circumstances:

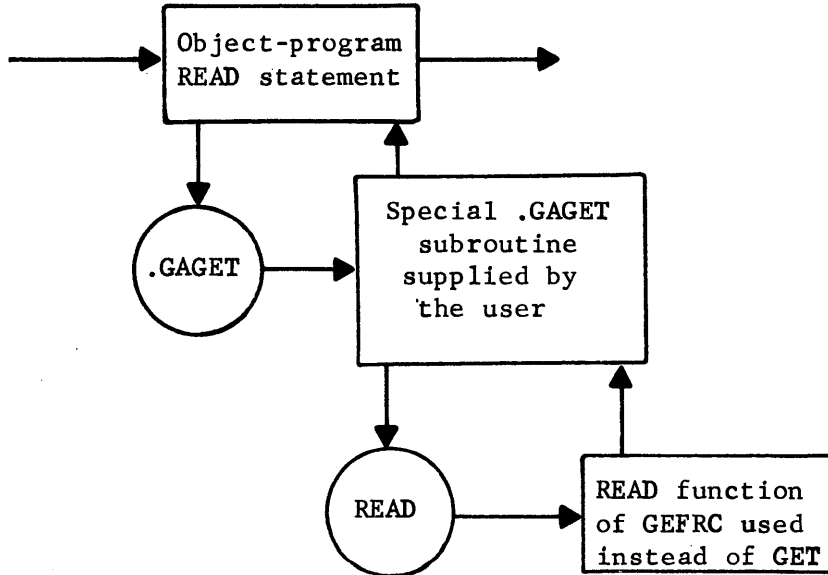
1. Normally, no special subroutine is present.



2. A simple format conversion subroutine supplied by the user might require GET.



3. A complicated stranger file format might require the special .GAGET subroutine to use GEFRC services other than GET; when the object-program "thinks" it is calling GET via the alternate symbol .GAGET it does not engage GET at all.



Utilization of Plug-in Points

The special plug-in subroutine is developed via GMAP as a relocatable binary subroutine, having as its entry symbol the appropriate GEFRC alternate entry symbol.

At object-time, GELOAD will link the object-program to the special subroutine instead of the GEFRC module intended by the compiler, provided the special subroutine appears in one of the following places:

1. With the object-program itself;
2. In a user library file of relocatable subroutines (such a file must be specified to GELOAD via a control card); or
3. In the system library in such a position that GELOAD's library search will encounter the special subroutine before encountering GEFRC modules.

Normally, of course, no special subroutine is present, and the GEFRC module is directly engaged instead.

Provided the target format requirements discussed previously are satisfied, the objectives specified in the introduction can be fully realized. The last objective, fully integrating stranger hardware into the COBOL object-program environment, is accomplished by placing the special subroutine in the system library.

Since the user-supplied subroutine overrides only the alternate entry symbol of a GEFRC module, the GEFRC module is still available via its standard entry symbol.

APPLICATIONS

File Conversion to GE-625/635 Formats

The plug-in capability has particular interest for converting files to GE-625/635 formats. Ideally, such a conversion would have the following properties:

1. No special pass on the data would be required--conversion would be a by-product of the first update cycle on the GE-625/635 computer.
2. No special version of the COBOL program would be required.
3. The COBOL-oriented applied programmer should direct his work to GE-625/635 formats, as if no conversion were necessary.

These ideal properties are realized in the following steps:

1. Plan the appropriate GE-625/635 format for the file.
2. Develop the "pure" COBOL source-program as if its files were in GE-625/635 format.
3. Develop via GMAP a relocatable binary subroutine, with entry symbol .GAGET to do the record format translation, record by record.
4. In the first GE-625/635 production cycle, submit the special .GAGET subroutine with the object-program for execution. Each time the object-program "thinks" it calls .GAGET the call is intercepted by the subroutine, and conversion proceeds, record by record. (If several input files are involved, tests in the subroutine must distinguish between them.)
5. Then discard the subroutine; its work is finished.

A proper GE-625/635 format file is thus produced in the first GE-625/635 update cycle.

Stranger Peripherals

The information needed to translate record formats for stranger peripherals is provided at the COBOL object-program interfaces with GEFRC.

Consider, for example, a customer who wishes to do all bulk card punching on a satellite computer. For reasons peculiar to the stranger machine, the standard GE-625/635 Bulk Media Conversion format cannot be utilized; records require "strange" control parameter formats, and record sizes cannot be aligned with GE-625/635 machine words.

A single special subroutine (symbol .GAWTR) is placed in the system library. It intercepts all .GAWTR calls; if "card" medium-code is specified in the calling sequence, it converts the record, while otherwise it simply passes the call on to WTREC. On card output records, it edits in suitable control parameters, exchanges the character set if necessary, and blocks the records in a form acceptable to the satellite computer, probably using the GEFRC WRITE function to deliver a full block at a time to magnetic tape.

Since the special .GAWTR subroutine is in the system library, it is automatically used by every object-program. COBOL object-programs are thus fully adapted to the (hypothetical) stranger satellite computer for card output files.

For another example, using a special .GAWTR subroutine to communicate with stranger printers, see T.I.B. number 600-3.

Recurrent Stranger Communications

Circumstances sometimes require that a certain file be produced for a stranger computer in each production cycle, or that a stranger file be accepted in each cycle. The technique utilized resembles that for file conversions to GE-625/635 format, except that the conversion subroutine is permanently appended to the object-program.

Summary

The above discussion explores only some of the most obvious applications for the plug-in capability. Such possibilities as random file processing, or object-time adaptation of files planned for tape to standard formats for linked mass storage files, are also introduced, presenting a spectrum whose scope will no doubt be explored as the need arises.

APPENDIX E
COBOL EFFICIENCY TECHNIQUES

INTRODUCTION

The following discussion presents techniques for achieving efficient GE-625/635 COBOL object-programs. Consideration is given to input/output, data manipulation, and data description techniques.

Some of the recommendations lead to space-saving, some to time-saving, and some to both. Each is flagged appropriately by (T), (S), or (T and S) to indicate the type of saving expected.

INPUT/OUTPUT

1. APPLY PROCESS AREA can save a considerable number of generated instructions if many statements refer to data items within a file. The memory saving may exceed the storage required for the PROCESS AREA. Especially important, however, may be the execution time saving if the verbs referring to a file's contents are heavily exercised. The implied MOVE between PROCESS AREA and buffer, supplied by GEFRC, is quite efficient in execution time. PROCESS AREA is especially recommended for a file containing OCCURS items. (T and S.)
2. The various files in a data processing program often fall naturally into high-volume and low-volume categories. For such a program, consider RESERVE ALTERNATE AREA for the high-volume files. This option permits the object-program's processing to overlap with input/output functions for such files. (T)
3. If two or more files will definitely not be open concurrently, use SAME AREA to conserve storage. This option causes the specified files to utilize the same buffer area(s). (S)
4. In a file maintenance program, consider SAME RECORD AREA for the "throughput" or master file. This option causes an implied PROCESS AREA to be applied to both the input and output versions of the file. (T)
5. If a PROCESS AREA is definitely not wanted for a throughput file, work on each record in the input buffer (where READ leaves it), and then transmit it to output via WRITE FROM. This approach simplifies insertion of new records. Such insertion requires extra housekeeping when SAME RECORD AREA is used. (S)
6. Both READ INTO and WRITE FROM imply MOVEs within core storage. Explicit or implicit PROCESS AREA also implies a MOVE on each READ or WRITE. Therefore, avoid the combination of PROCESS AREA

with READ INTO or WRITE FROM. For example, the worst possible inefficiency would result from having a PROCESS AREA for both the input and output master files, and then using READ INTO a Working-Storage record area and WRITE FROM that area:

- a. Because of the PROCESS AREA, the READ implies a MOVE from buffer to PROCESS AREA.
- b. The INTO option implies a MOVE from PROCESS AREA to Working-Storage (in this example).
- c. The FROM option of WRITE implies a MOVE from Working-Storage to the output file's PROCESS AREA (in this example).
- d. Because of the PROCESS AREA, the WRITE implies a MOVE from PROCESS AREA to output buffer.

Four MOVES here have done the work of one! (T and S)

DATA MANIPULATION

1. Avoid the CORRESPONDING option when a simple MOVE would suffice. MOVE CORRESPONDING results in a series of MOVES of individual items; a simple MOVE is instead optimized for the group or record as a whole. Never use MOVE CORRESPONDING for such purposes as transmitting a master file record from the input buffer to the output buffer. Use MOVE CORRESPONDING when it will in fact cause selected items to be MOVED, or when editing or format conversion is needed on the respective items. (T and S)
2. Manipulate a group or record as a whole (whenever possible), rather than manipulating its elementary items separately. This rule is especially important for tables of data items--MOVE or clear a table as a whole whenever possible. For example, technique a. (below) is quite efficient, while b. is terribly inefficient:
 - a. MOVE SPACES TO TABLE.
 - b. COMPUTE I = 1.
LOOP. MOVE SPACES TO TABLE-ITEM (I).
COMPUTE I = I + 1.
IF I NOT > TABLE-SIZE GO TO LOOP.(T and S)
3. If a subscripted item is to be referred to more than once with the same subscript value(s), consider MOVING it to a temporary Working-Storage area once for all. (T and S)
4. If a data item is to be used in several subscripts without a change in value, either make it a COMPUTATIONAL-1 item or else MOVE it to a temporary area in Working-Storage (described as COMPUTATIONAL-1) and use the Working-Storage item in the subscripts. (T and S)

5. For MOVES, conditions, addition, and subtraction, give the items like PICTURES and USAGES whenever possible. (T)
6. In the UNTIL option of PERFORM, use the simplest possible condition to terminate the loop. If necessary, achieve such simplicity by preceding the PERFORM with explicit MOVES and COMPUTEs. If NUMERIC items are involved in the condition, give them like PICTURES and the same COMPUTATION (-1 or -2) USAGE, not DISPLAY USAGE. (T)
7. Tend to utilize procedural literals, rather than Constant Section items or constant values in Working-Storage. The compiler can optimize the format and word-orientation of procedural literals, but must resort to dynamic format conversions in the object-program if Constant or Working-Storage items are not ideally aligned. (The user can, of course, accomplish ideal alignment by using exceptional care in his data description--but this practice is not recommended for ordinary constant values. Use of Constant Section items when doing so will enhance the self-documentation or maintainability of the program; otherwise use procedural literals.) (T)
8. Use GO...DEPENDING for decisions whenever possible. In any situation for which GO...DEPENDING can be used, it yields considerably more efficient object-coding than a succession of IF statements. (T and S)

DATA DESCRIPTIONS

1. Whenever possible, use COMPUTATIONAL(-n) USAGE for a NUMERIC item which will be involved in formulas, arithmetic verb statements, or numeric comparisons. External considerations sometimes dictate DISPLAY USAGE; in such case, consider explicitly MOVING the item to a COMPUTATIONAL(-n) Working-Storage area once for all processing. (Such a MOVE would of course be inappropriate if only one procedural statement refers to the item.) (T)
2. COMPUTATIONAL(-n) USAGES should be employed in the following preferential order:
 - a. Use COMPUTATIONAL-1 if the item is an integer and is not involved arithmetically with COMPUTATIONAL or COMPUTATIONAL-2 items. (T)
 - b. Use COMPUTATIONAL if the item is not an integer or if it plays off with other COMPUTATIONAL items. Be sure to use COMPUTATIONAL if precise fractional results are required. (T)
 - c. Use COMPUTATIONAL-2 if the advantages of binary floating point are needed. (The main advantage is capacity for accurate representation of very large or very small numeric values.) Arithmetic coding for COMPUTATIONAL-2 items is highly efficient, but it does not yield the exact decimal results needed in most commercial applications. (T)

3. Specify COMPUTATIONAL-1 for an item which will be used as a subscript or which will be a DEPENDING item in a GO statement or in an OCCURS clause. This rule is especially important if the item will be mentioned as a "subscript-name" in PERFORM...VARYING, or in any such loop. Again, consider MOVING the item explicitly to a COMPUTATIONAL-1 area in Working-Storage if other considerations dictate DISPLAY USAGE. (T)
4. For data items in Working-Storage which do not specifically require close packing across computer words, specify SYNCHRONIZED. This rule is especially important for repeated items in a table (OCCURS), and most of all in any table which must be repeatedly searched. (T and S)
5. If a record contains COMPUTATIONAL(-n) and/or SYNCHRONIZED items, put single-word items together and put double-word items together whenever possible. Considerable storage economy can result; this rule is clearly most important for records in a file. (S)
6. For COMPUTATIONAL(-n) items, make the PICTURE clause single-word precision whenever it suffices, rather than double-word precision. (S)
7. If the end of a record does not coincide with the end of a word, implied filler accounts for the unused character positions. Tend to specify explicit FILLER for this purpose, rather than allowing implicit filler. (In doing so, however, note that the result is specifically oriented to the GE-625/635 computer word size.) (T)
8. Consider giving all records in a VLR file an odd word-length; in a FLR file, even word-length. (T)
9. It is often necessary to organize peripheral files in a highly space-saving manner, even though time-saving during processing is also greatly desired. In such a case, describe each record both in the File Section and in Working-Storage. In the File Section, pack the data as closely as possible, without regard to processing efficiency; in Working-Storage, do exactly the opposite. Avoid PROCESS AREA (if possible) and avoid READ INTO and WRITE FROM. Instead, READ each record, and, while it is still in the input buffer, determine whether a transaction applies to it (or, in general, whether it must become involved in detailed processing). If detailed processing is needed, employ MOVE CORRESPONDING to unpack either the entire record or the interesting group(s) within it to the Working-Storage area and refer to the data there for all detailed processing. Similarly, employ MOVE CORRESPONDING as appropriate to construct (or reconstruct) the output record. Do a simple MOVE from input buffer to output buffer if detailed processing is not required. (T and S)
10. If reporting is done without Report Writer, use skeleton lines in Working-Storage, with constant information initialized via VALUE rather than via MOVE statements in the Procedure Division. (T and S)

APPENDIX F
COBOL REPORT WRITER OBJECT MODEL

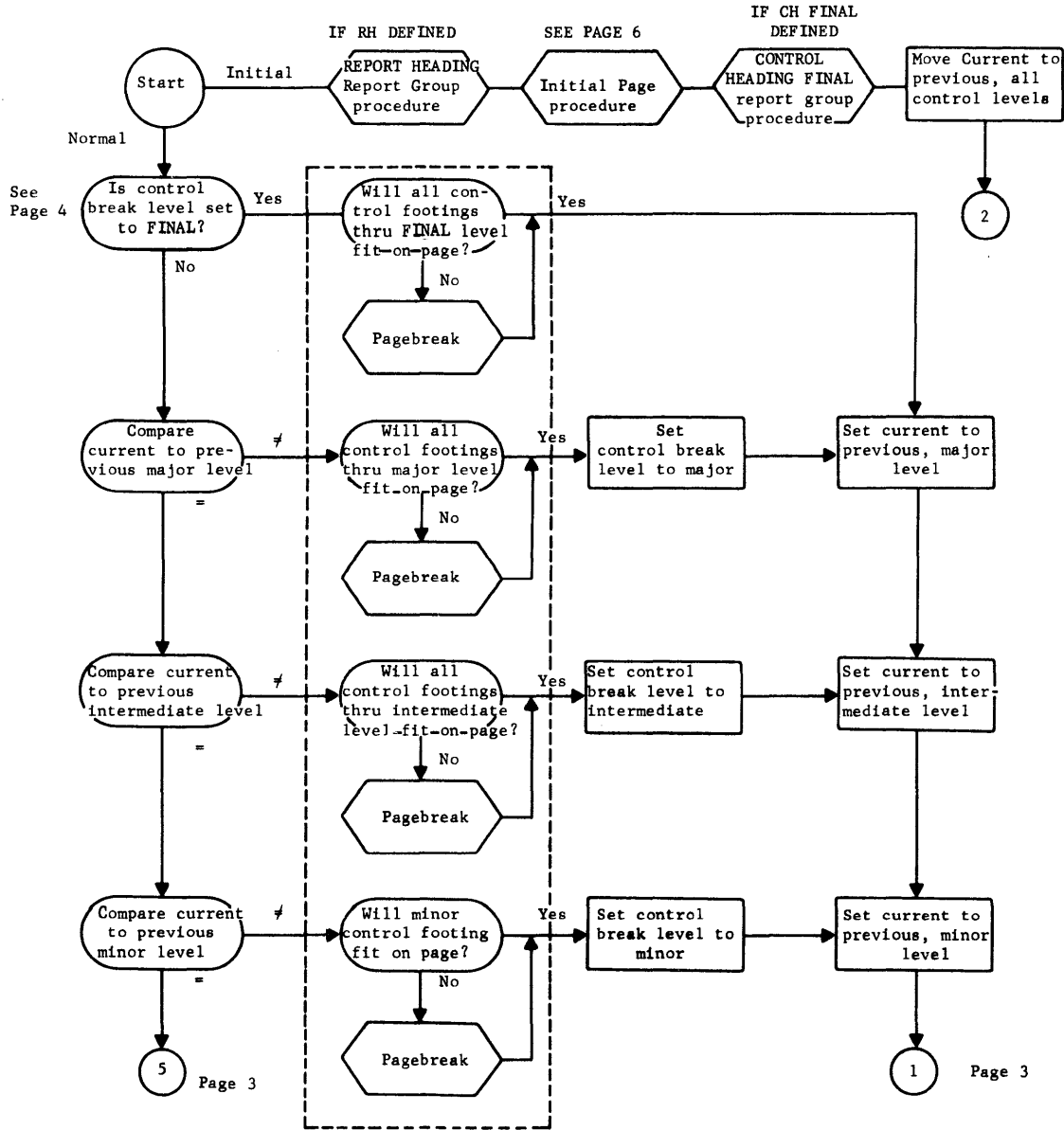
This Appendix presents schematic flowcharts of the object-program procedures generated for a report. The flowcharts should answer most questions about control key manipulation, USE BEFORE REPORTING, rollforward of totals, PAGE LIMITS, and similar considerations. However, they omit logic related to rule violation and recovery and omit detailed logic for establishment of PAGE versus OVERFLOW condition, GROUP INDICATE, actual line editing, SUM accumulation, etc.

The GENERATE flowchart reflects three control levels, an associated TYPE CONTROL FOOTING, and TYPE CONTROL HEADING report group for each level. The following definitions apply:

current = current value of a CONTROL data name.

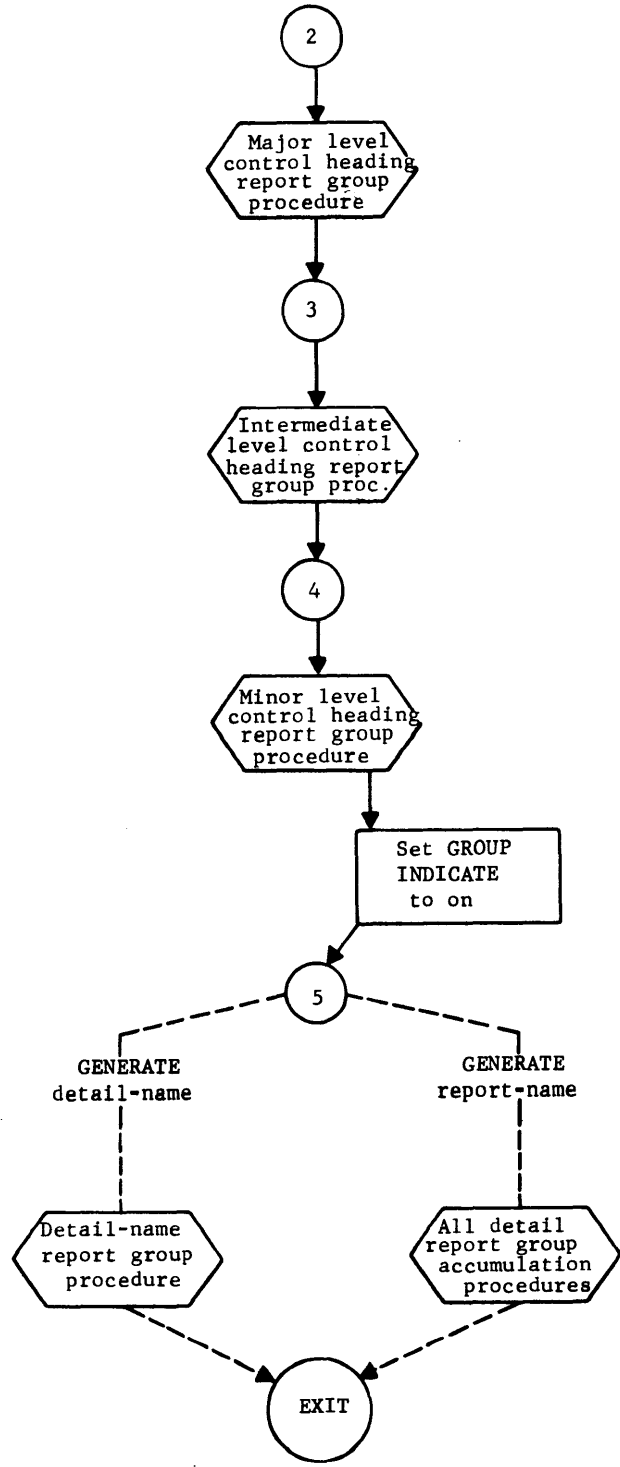
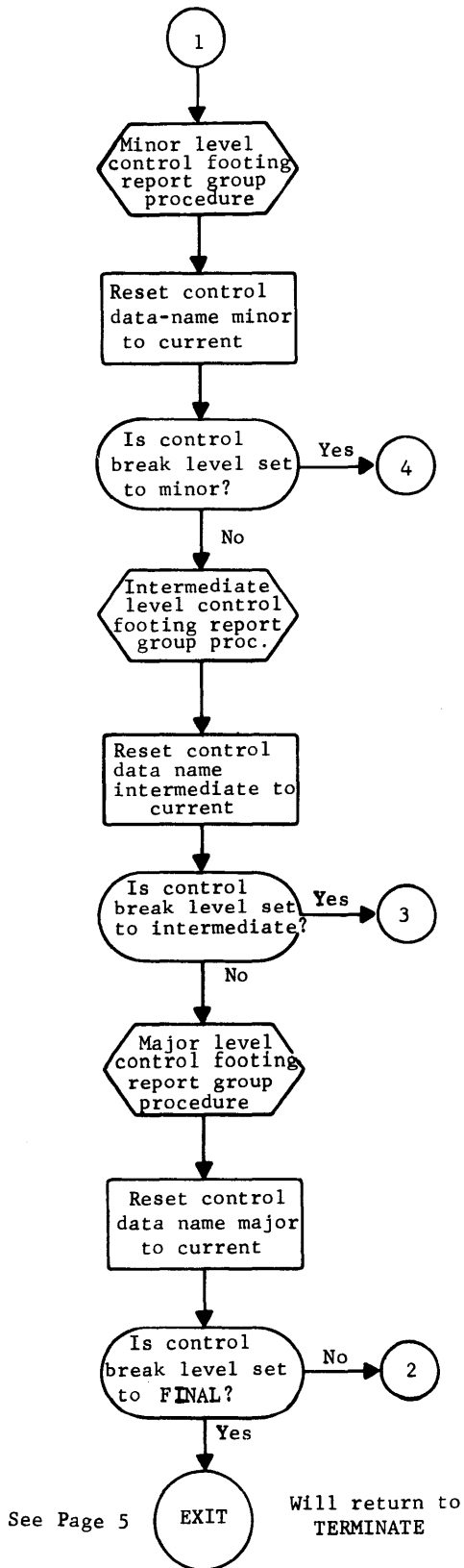
previous = the value of a CONTROL data name at previous GENERATE execution.

GENERATE
(overall logic only)



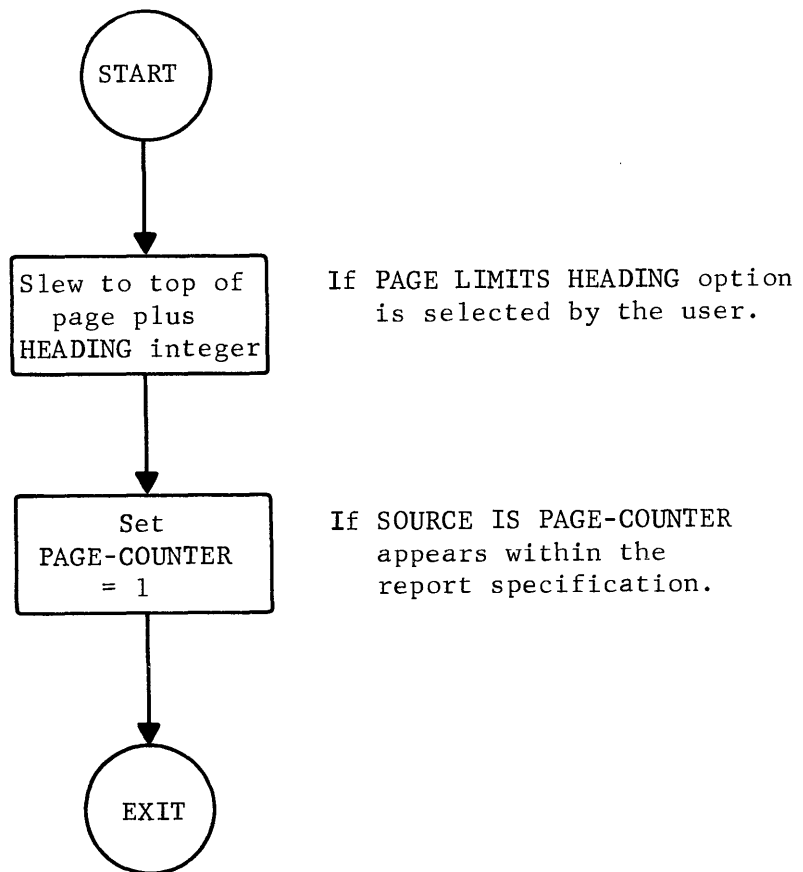
Note: The area in the hatched rectangle is included only if the PAGE LIMITS FOOTING option is selected by the user.

GENERATE (continued)

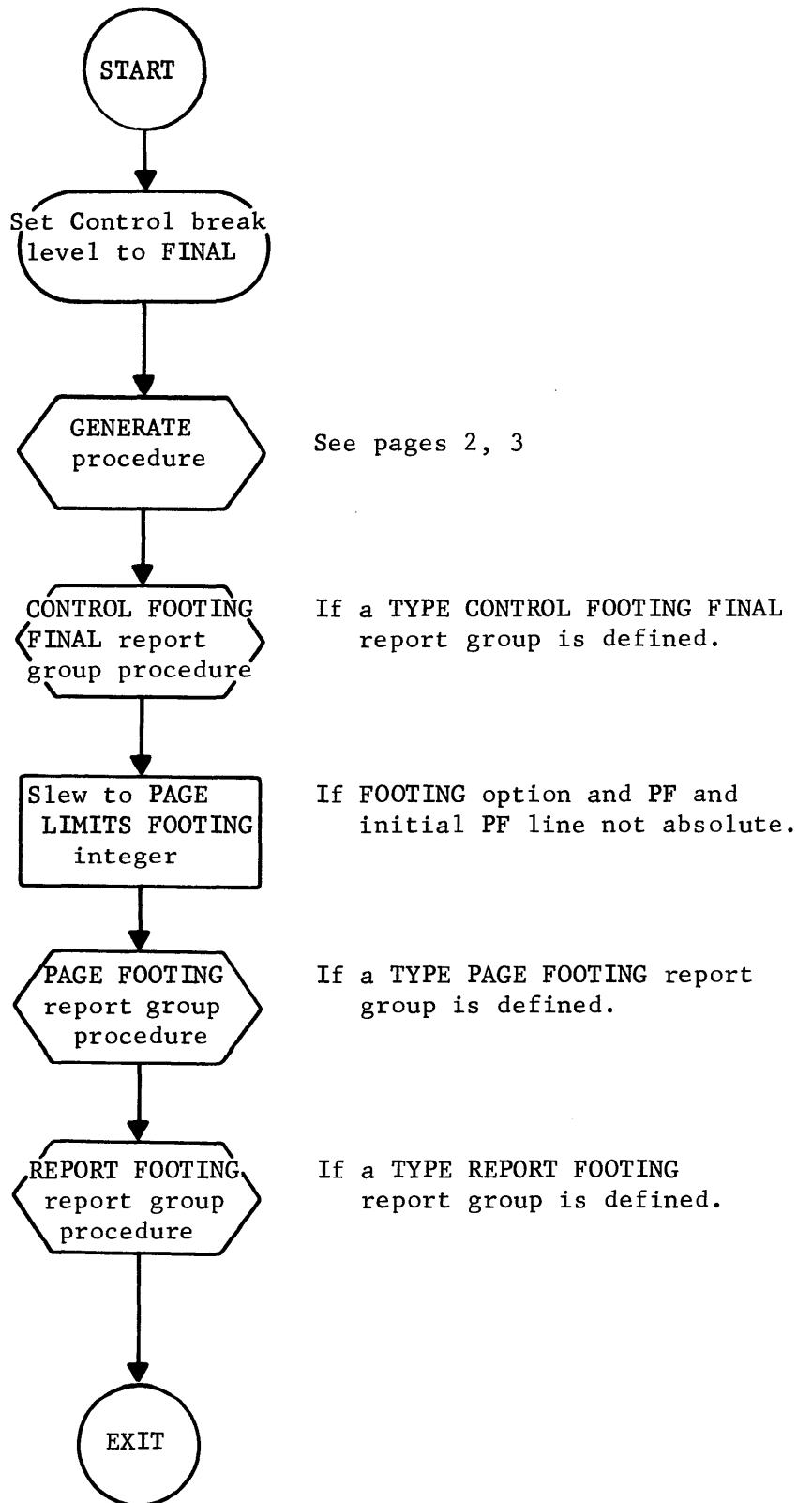


See page 7 for outline of report group procedures.

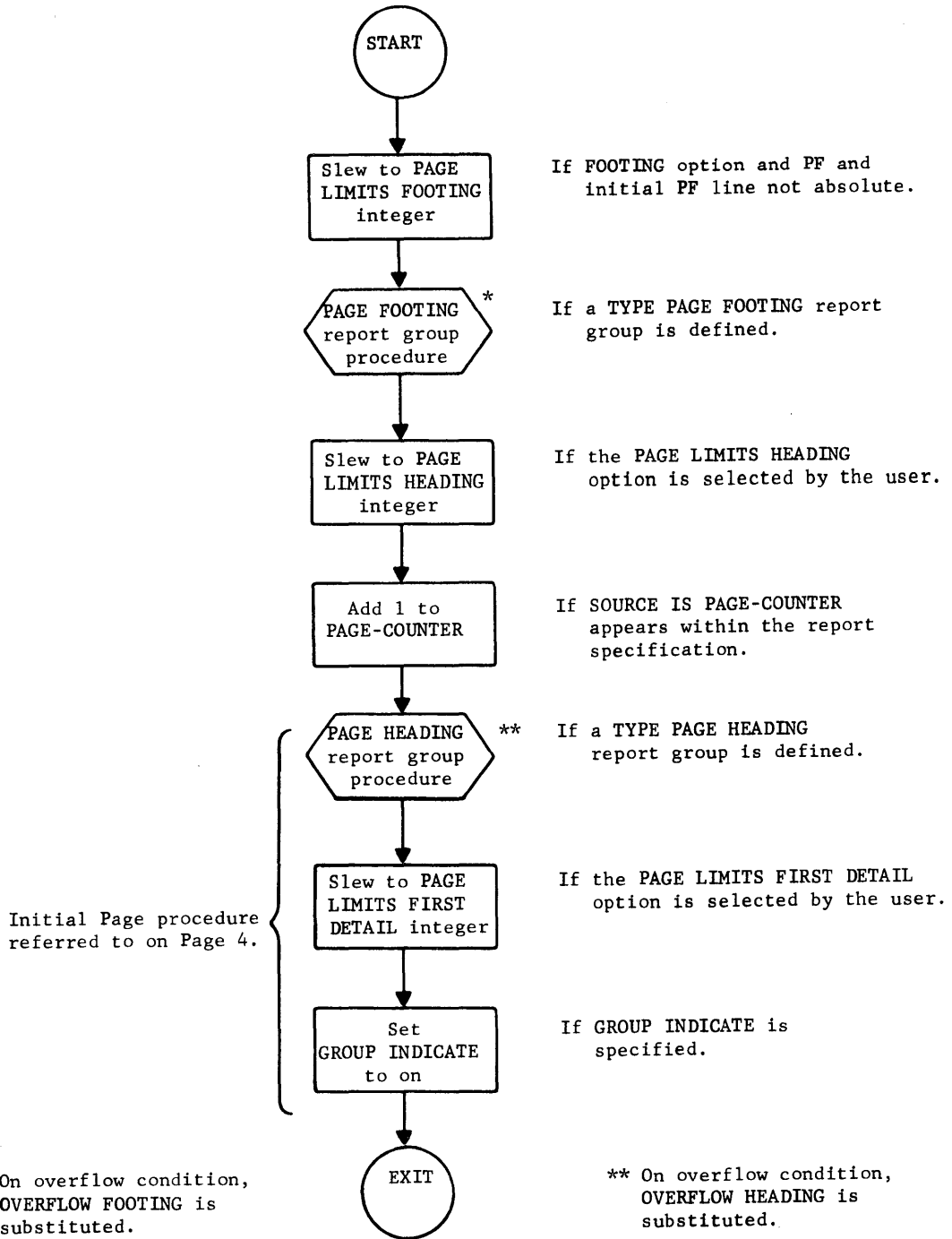
INITIATE
(overall logic only)



TERMINATE
(overall logic only)



PAGE BREAK



REPORT GROUP PROCEDURES
(in order of execution)

All functions are conditionally generated, based on report parameters.
The procedures are as follows:

1. Page limit testing (If at limit, Pagebreak is executed);
2. USE BEFORE REPORTING section;
3. Output of line printing and/or slew records, including GROUP INDICATE when appropriate;
4. Accumulation of current detail or sum-counter values to sum-counters;
5. Reset sum-counters to zero if TYPE CF.

APPENDIX G
COBOL COMPILATIONS

Two features have been added to COBOL. These are described in the following paragraphs:

AVAILABILITY OF COMPRESSED SOURCE DECKS

A compressed source deck (COMDK) may be requested in the \$ COBOL control card. This is done by use of COMDK as an option in the operand field of the \$ COBOL control card. The standard option, NCOMDK, is assumed when COMDK is not used.

These two options are summarized as follows:

NCOMDK - No compressed deck of the source program will be prepared.

COMDK - A compressed deck of the source program will be prepared.

FASTER COMPILATIONS

Faster compilations are obtained for source decks which do not use either the COPY clause in the Data Division or the file RENAMING clause in the Environment Division.

It is necessary for all source decks which use the COPY clause or the file RENAMING option to specify COPY as an option in the operand field of the \$ COBOL control card. The standard option, NCOPY, is assumed when COPY is not specified.

The two options are as follows:

NCOPY - No COPY prepass is made on the source program.

COPY - A COPY prepass is made on the source program to process any COPY clauses or file RENAMING clauses specified in the source program.

For additional information on the use of the \$ COBOL control card, see GE-625/635 Comprehensive Operating Supervisor, CPB-1195.

APPENDIX H
SQUEEZE-CODING IN COBOL SORTS

The COBOL library subroutine .CSORT permits COBOL programs to use squeeze-coding routines. These are used for analysis and conditional record-deletion of records when equal key values are encountered during sorting.

GENERAL DESCRIPTION

The label .CSRTE (in the library routine .CSORT) is a SYMDEF which permits the address of a squeeze-code routine to be placed into .CSRTE+15. The address must go into the lower half of .CSRTE+15 without disturbing the contents of the upper half.

The coding used to enter the squeeze-code routine must be in GE-625/635 Macro Assembly Program (GMAP) language and must be executed before execution of the SORT statement. The address of the squeeze-code routine (in .CSRTE+15) is left unchanged by the sort program. If this address is not changed by the user, the same squeeze-code routine is used whenever a SORT statement is executed.

Changing Squeeze-Code Routines

The selection of the squeeze-code routine can be changed by changing the address in location .CSRTE+15. This must be done between SORT executions-- after the completion of one SORT and before the execution of the next.

Discontinuing Squeeze Code Routines

Resetting the lower half of location .CSRTE+15 to zero causes no further squeeze-code routines to be executed. This "turning off" of the squeeze must be done between sorts and not during execution.

SQUEEZE-CODE ROUTINES

The user must write his squeeze-code routines in the GMAP assembly language. Index registers 6 and 7 must refer to the records having equal key values. The routines must protect the contents of all registers for the sort program. The conventions for deleting records are as described in the GE-625/635 Sort/Merge Program reference manual, CPB-1005.

EXTRA CODING REQUIRED

Extra coding required in the COBOL program consists of the following three elements:

1. Coding in GMAP assembly language to set the squeeze-code address into .CSRTE+15.
2. Assembly language coding to change the address of the squeeze-code routine from one routine to another or to reset .CSRTE+15 to zero.
3. Assembly language coding of the squeeze-code routine (s).

APPENDIX I
OPTIONAL COMPILATION OF
STATEMENTS IN GE-625/635
COBOL SOURCE PROGRAMS

GENERAL DESCRIPTION

Standard source statements can be selectively compiled or bypassed at the request of the programmer.

The programmer designates those statements which are to utilize this capability by placing one of the digits 0-9 in Column 7 of the source cards. The request to compile or bypass the statements on subsequent compilation runs is then communicated to the compiler by means of an entry in the SPECIAL-NAMES paragraph. (See Page VIII-4.)

FUNCTION

The optional compilation of specified source statements can be used effectively in both the debug and production phases to improve overall program flexibility.

Debug Phase

A considerable amount of checkout time and effort can be saved by using standard COBOL source statements to trace program errors, rather than using inconvenient octal core or tape dumps.

Special counters can be used to record test information and the results displayed on-line. Intermediate results or the status of selected records before and after processing can be written to SYSOUT or to a special test-file. Literal values can be moved to data-items and logical sequences or computational results checked. The actual sequence of program execution can also be traced by placing DISPLAY statements at strategic points. In this way, most of the program checkout can be done at the source language level. In addition, the DEBUG statements can be grouped, by means of a common special identifier in Column 7, to provide various levels of detail on successive checkout runs. When the checkout phase is completed, only the SPECIAL-NAMES request card need be removed, leaving all of the test statements in the source deck in an inactive status.

Production Phase

Special purpose capabilities within a basic program can be written with optional statements specifying the variations. In this way a single source program configuration can be maintained and year-end, field-site or other special-purpose routines activated by requesting compilation of the appropriate statements as needed.

METHOD OF IMPLEMENTATION

Column 7 Identifier

Column 7 of the source statement card is used to identify an entry which may be conditionally compiled. Any of the digits 0 through 9 can be used for this purpose but normally a single digit will be used to identify groups of statements used for different purposes.

SPECIAL-NAMES Paragraph Entry

To indicate to the compiler which of the specially identified (Column 7) statements are to be compiled, one of the following options must be requested in the SPECIAL-NAMES paragraph:

Option 6:

SPECIAL-NAMES.

PROCESS ALL DEBUG STATEMENTS.

Option 7:

SPECIAL-NAMES.

PROCESS LEVEL integer-1 [THRU integer-2]

DEBUG STATEMENTS.

Notes:

1. Option 6 is used when all statements identified by 0-9 in Column 7 are to be compiled.
2. Option 7 is used when it is desired to compile either a single group of statements (identified by integer-1), or a range of groups (identified by integer-1 THRU integer-2).
3. If neither Option 6 nor Option 7 is selected, then all statements identified by a 0-9 in Column 7 will be unconditionally bypassed during compilation.

GENERAL USAGE NOTES

1. It should be clearly understood that this feature is unique to GE-625/635 COBOL. Consequently, any program utilizing it may have to be modified before being compiled on another computer line.
2. This facility may be used freely in the Input-Output Section of the Environment Division, Data or Procedure Divisions, except any entry or statement with a COPY clause.

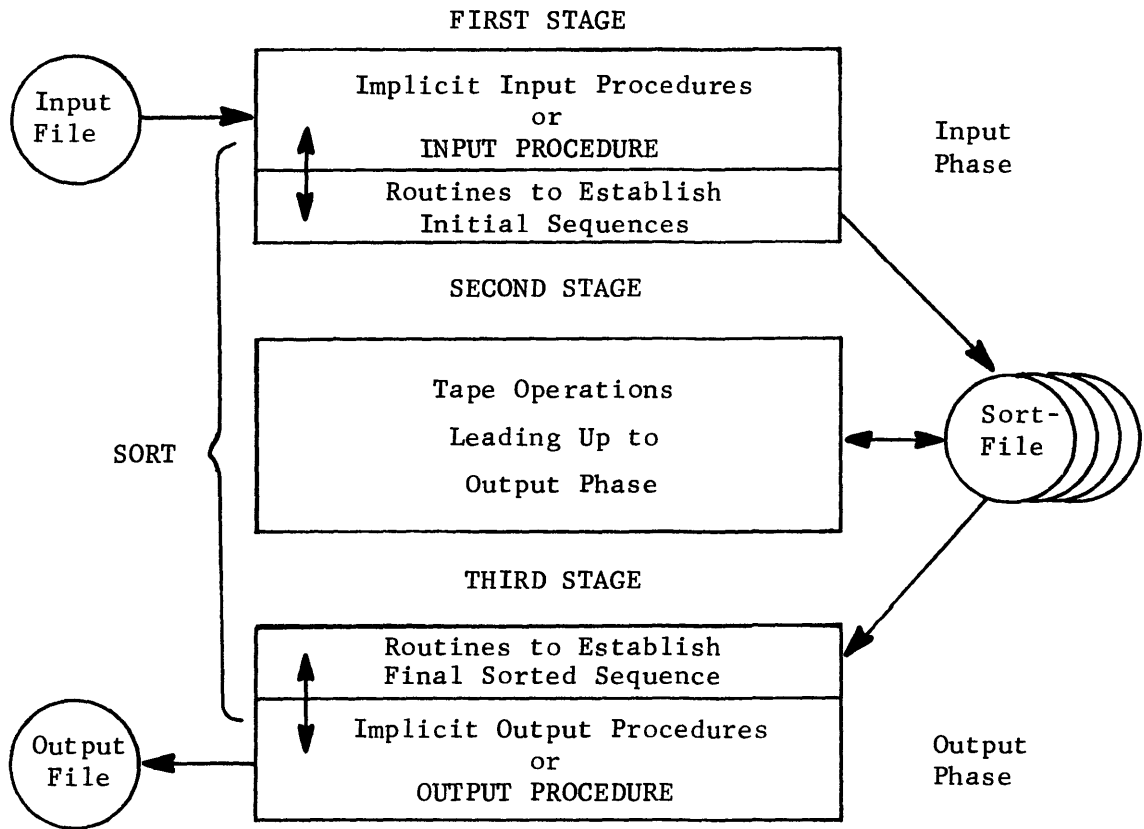
3. Only source statements containing one of the digits 0-9 in Column 7 are affected by this special clause. The use or omission of the appropriate options in the SPECIAL-NAMES paragraph has no affect on any source statement with a blank, hyphen or asterisk in Column 7.
4. Do not attempt continuation of an optional compilation statement to a second line by means of a hyphen in Column 7. These requirements are mutually exclusive.

APPENDIX-J

USE OF THE COBOL SORT FEATURE

SORT PROGRAM ORGANIZATION

The Sort program is organized to cope with potentially quite large amounts of data, exceeding many times over the capacity of the computer's internal memory. It utilizes magnetic tapes for this purpose. The sorting operation proceeds in three stages, as illustrated in the following chart:



In the first stage, the Sort accepts all of the data records to be sorted, rearranging them into sequences of as many records as memory will allow. (Typically, the initial sequences contain about twice as many records as can be held in available memory at once.) The initial sequences, called "strings," are distributed in a dynamically planned fashion over the available work tapes (called "collation tapes").

In the second stage, a series of merging operations combine the initial strings into longer strings, reducing the total number of strings outstanding. At the conclusion of this stage, merging has proceeded to the point that each of the several collation tapes has one long string of records.

In the third stage, a final merging operation combines the data into one string consisting of the entire file, rearranged now into the stipulated output order.

In the simplest application, the first stage of a Sort program reads the input file from magnetic tape, building the initial strings as it goes; and the third stage of the Sort writes the output file to magnetic tape.

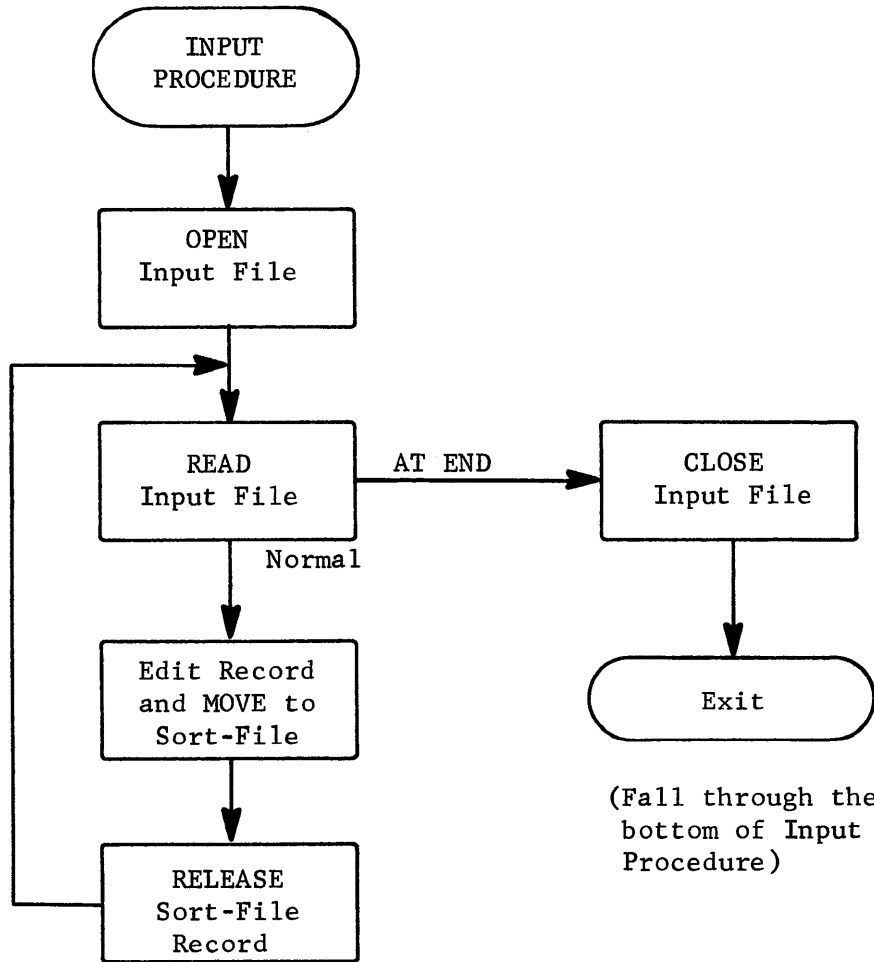
In a more complicated application, the user may optionally set aside the normal reading function of the Sort and supply in its stead an Input Procedure tailored to his own needs. The usual purpose of such a procedure is to preprocess the data records, editing, and perhaps dropping information not needed in the output file produced by the Sort.

Similarly, the user may optionally set aside the normal writing function of the Sort's third stage, supplying in its place an Output Procedure adapted to his needs. The normal use of such a procedure is to edit the data records and produce a report directly in the last stage of the Sort, rather than letting the Sort produce an output file which must become input to a subsequent reporting program. When it is feasible, this use of an Output Procedure saves a complete pass of the data through the computer.

RESULT OF EXECUTING A SORT STATEMENT

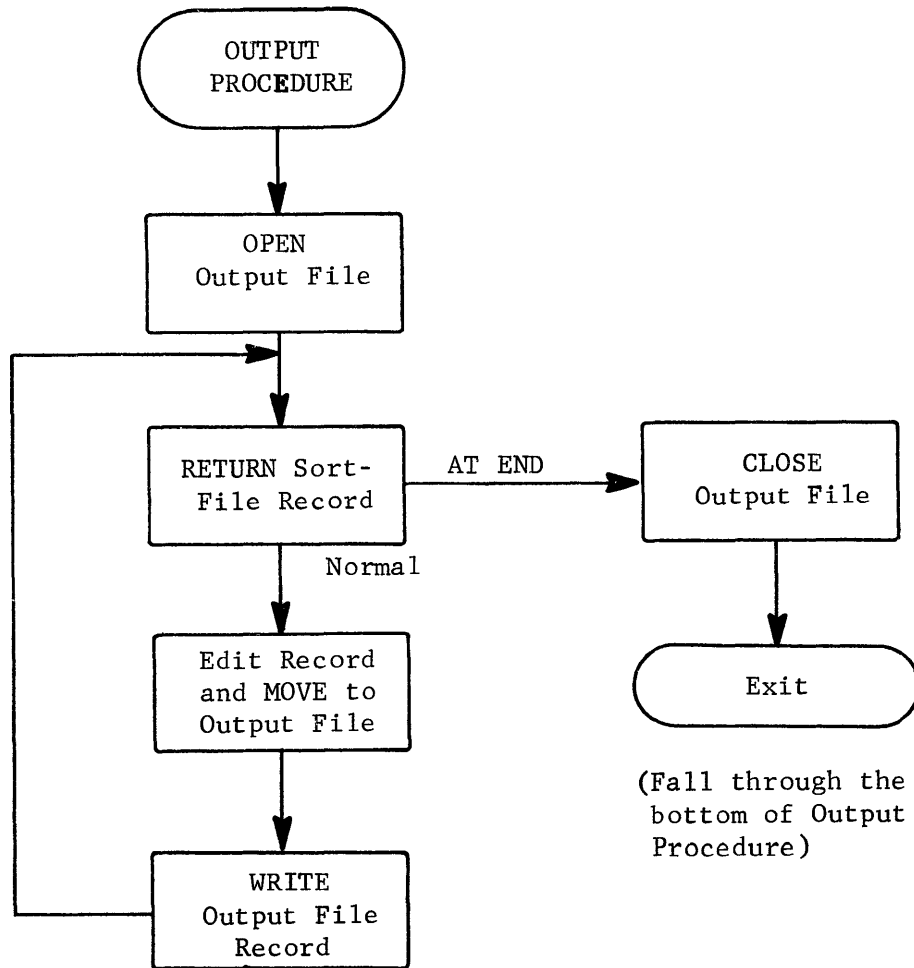
A close look at the sequence of events in the object program should enhance an understanding of the COBOL rules. Consider an object program containing a SORT statement. After the object program has been loaded, its execution begins with the first nondeclarative procedural statement, as usual. The object program may then execute any sequence of procedural statements leading up to the SORT statement. The SORT statement immediately gives control to the Sort subprogram.

The Sort does its initial housekeeping; and, assuming an Input Procedure has been specified, it transfers control to the Input Procedure as soon as it is ready to start processing records. The Input Procedure may now accomplish any necessary initialization, after which it opens its input file and reads the first record. When it has a record ready to enter the Sort, the Input Procedure submits it to the Sort by executing a RELEASE statement. The Sort then processes the record for positioning in one of the initial sort-file strings, and returns control to the statement following the RELEASE statement. The Input Procedure continues reading and releasing until all the input records have been given to the Sort, at which time the Input Procedure does its final operations including closing its input file. The Input Procedure signals an end-of-input condition to the Sort by allowing control to pass to the end of the Input Procedure. The Sort then concludes the first stage of processing and enters the second stage. A simple Input Procedure might thus be organized as shown on the following page.



The second stage of the Sort is a merging operation which has no direct interface with the COBOL program. When its function has been accomplished, the second stage passes control to the final stage.

The final stage of the Sort does its initial housekeeping, up to determining which record goes first in the final output sequence. Assuming an Output Procedure has been specified, the Sort at this point transfers control to the Output Procedure. The Output Procedure may begin with any necessary initialization, including opening its output file. It then obtains the first record in the final sequence by executing a RETURN statement. When it has processed and written or otherwise disposed of the first record, it returns the next, and thus it continues returning records and processing them to output. The RETURN verb has the same AT END provision as the READ verb's, so eventually the Sort makes a special exit to the AT END procedure on the first RETURN statement executed after the last data record has been returned. The Output Procedure concludes its operations, closing its output file, and finally allows control to pass to the end of the Output Procedure. Control then returns to the Sort, which terminates its own procedures and returns control to the source program statement following the SORT statement. A simple Output Procedure might thus be organized as shown on the following page.



(Fall through the bottom of Output Procedure)

In effect, the sequence of events just described applies also when the USING or GIVING option is used, except that the Input or Output Procedure becomes implicit, not specified in detail by the user or generated by COBOL.

The user can, in principle, include several SORT statements in the source program, even embedding them in loops. If several SORT statements are present, they are completely independent of each other.

FILES RELATED TO COBOL SORT VERB

A SORT statement may be resolved into the following four elements:

1. The Sort file-name is specified first. The only physical realization of this "file" is the set of collation tapes used by the Sort.
2. The key data-names are then specified. (Keys are not further considered here.)
3. Either an Input Procedure or a USING file-name must be specified, to define the input data source for the Sort.
4. Either an Output Procedure or a GIVING file-name must be specified, to define the output data destination for the Sort.

The Input Source

If the USING option is specified, the input file must be described via an FD. It must be assigned a unique file code in a SELECT sentence, and must be represented by an appropriate \$ FILE card when the object program is executed.

If an Input Procedure is specified, the user assumes responsibility to obtain input records for the Sort. Required source program statements and control information are accordingly governed by ordinary input file requirements.

The input file code is not suitable for collation tape use.

The Output Destination

If the GIVING option is specified, the output file must be described via an FD. It must be assigned a unique file code in a SELECT sentence, and must be represented by an appropriate \$ FILE card when the object program is executed.

If both USING and GIVING options are specified, they may refer to the same file-name or to different file-names. In the special case in which USING and GIVING refer to the same file-name, the Sort will not use the GIVING file medium as a collation tape. Otherwise, the Sort will indeed use the GIVING file medium as a collation tape, so the actual output medium must then be magnetic tape.

If an Output Procedure is specified, the user assumes responsibility to deliver the sorted data to output. Required source program statements and control information are accordingly governed by ordinary output file requirements. If the actual output medium is magnetic tape, the tape may be supplied to the Sort for use as a collation tape by a method described below.

The Sort-File

The sort-file must be described via an SD. It must be assigned one or more file codes in a SELECT sentence. Normally, only a single file code will be specified. The first (or only) file code must be unique.

The programmer controls the disposition of collation tapes by his use of file codes for the sort-file. The following presents several alternative file code strategies, and assumes that the reader understands the \$ TAPE and \$ NTAPE control cards of GECOS.

A basic requirement is that the Sort must have at least three collation tapes. Whichever file code strategy is chosen, this need must be satisfied. Sources of collation tapes are as follows:

1. A \$ NTAPE card specifying file code S1 may provide some or all of the collation tapes. Depending upon how many tapes the \$ NTAPE card provides, file codes in the series S1, S2, ... will be implicitly available for the Sort. Because of their special use for Sort, file codes in this series must not be explicitly mentioned in any COBOL program which utilizes Sort.

2. As described on the preceding page (see The Output Destination) the GIVING option normally provides one collation tape to the Sort when the GIVING option is specified.
3. Additional collation tapes may be provided via extra file codes specified in the sort-file SELECT sentence.

It is recommended that the first (or only) file code mentioned in the sort-file SELECT sentence not be represented by a \$ TAPE card when the object program is executed. Furthermore, the \$ NTAPE card approach is considered the standard means of providing collation tapes.

The typical file code strategy recommended for a sort-file is therefore to assign a single file code (thereby satisfying COBOL syntax rules); to omit the \$ TAPE card for that file code when the object program is executed; and to supply instead a \$ NTAPE card specifying file code S1.

Use of multiple file codes in the sort-file SELECT sentence may be desired for various reasons:

1. If an Output Procedure produces a magnetic tape output file, the file code assigned to that file should be the second mentioned in the sort-file SELECT sentence (and the first file code should definitely not be represented by a \$ TAPE card at execution time). This provision puts the tape at the Sort's disposal throughout the collation activity, but guarantees that the tape will be free by the time the Output Procedure is engaged.
2. Perhaps one or more tape files are totally processed before or after the Sort procedure, so that the tapes are indeed available as scratch tapes throughout the Sort execution. Listing their file codes in the sort-file SELECT sentence leads to enhanced sorting efficiency. Such file codes would also be mentioned in the SELECT sentences for the files to which they are assigned.
3. If for some reason the \$ NTAPE card is not desired, multiple file codes in the sort-file SELECT sentence can be employed to give the user full control over the allocation of collation tapes via \$ TAPE cards.

Provided the basic three collation tape requirement is somehow satisfied, \$ TAPE cards for the extra sort-file file codes may optionally be omitted at execution time. If \$ TAPE cards are provided, the Sort definitely assumes such tapes are available for collation purposes.

The USING and GIVING file codes must not be mentioned in the SELECT sentence for the sort-file.

When extra sort-file file codes are actually represented by \$ TAPE cards at execution time, each such file code counts as only one tape toward the Sort's minimum three collation tape requirements, even if the \$ TAPE card calls for alternating tape handlers. Similarly, the GIVING file code (when applicable) counts as only a single collation tape.

Examples

The following example illustrates a basic SORT utilization.

```

.
.
.
.
ENVIRONMENT DIVISION.
FILE-CONTROL.
    SELECT INPUT-FILE ASSIGN TO AB.
    SELECT SORT-FILE ASSIGN TO CD.
    SELECT OUTPUT-FILE ASSIGN TO EF.
DATA DIVISION.
FILE SECTION.
FD INPUT-FILE...
    01...
.
.
SD SORT-FILE...
    01...
.
.
FD OUTPUT-FILE...
    01...
.
.
PROCEDURE DIVISION.
SORT-CALL. SORT SORT-FILE ON ASCENDING KEY...USING INPUT-FILE
    GIVING OUTPUT-FILE.
STOP RUN.
END OF PROGRAM.
```

Note that these files may have multiple record types and sizes, provided the sort-file and USING file have the same records, the sort-file and GIVING file have the same records, and key descriptions and positions are equivalent for all record types.

At execution time, the control cards should be as follows:

```
.
.
.
$ EXECUTE
$ LIMITS ...
$ TAPE AB,A1D,,,,INPUT-LABEL
$ TAPE EF,B1S,,,,OUTPUT-LABEL
$ NTAPE S1,C,2 ← (2 or more tapes required
                  in this example)
```

Another important SORT utilization entails use of an Output Procedure to deliver a report (on any suitable device) rather than an output tape as such.

```
.
.
.
.
ENVIRONMENT DIVISION.
FILE-CONTROL.
    SELECT INPUT-FILE ASSIGN TO GH.
    SELECT SORT-FILE ASSIGN TO IJ, KL.
    SELECT REPORT-OUTPUT ASSIGN TO KL FOR LISTING.
DATA DIVISION.
FILE SECTION.
FD INPUT-FILE...
    01...
.
.
.
SD SORT-FILE...
    01...
.
.
.
FD REPORT-OUTPUT; REPORT IS XYZ...
.
.
.
REPORT SECTION.
RD XYZ...
    01 DETAIL-LINE; TYPE DE...
.
.
.
```

```

PROCEDURE DIVISION.
SORT-CALL SECTION.
DRIVER.  SORT SORT-FILE ON ASCENDING KEY...USING INPUT-FILE
        OUTPUT PROCEDURE IS EDIT.  STOP RUN.
EDIT SECTION.
STARTUP.  OPEN REPORT-OUTPUT; INITIATE XYZ.
LOOP.  RETURN SORT-FILE RECORD; AT END GO TO QUIT.
        GENERATE DETAIL-LINE; GO TO LOOP.
QUIT.  TERMINATE XYZ; CLOSE REPORT-OUTPUT.
END OF PROGRAM.

```

Because file code KL is used for the Output Procedure's output file and also is mentioned in the sort-file SELECT sentence, it must be associated with a magnetic tape at execution time. The control cards should therefore be as follows (assuming Sort is to use only three collation tapes):

```

.
.
.
$ EXECUTE
$ LIMITS ...
$ TAPE GH,A1D,,,,INPUT-LABEL
$ TAPE KL,B1S,,,,OUTPUT-LABEL
$ NTAPE S1,C,2

```

If KL were not mentioned in the sort-file SELECT sentence, its control card could specify any suitable output medium, such as \$ TAPE, \$ DISC, or \$ SYSOUT. Except in the latter case the Sort activity would presumably be followed by a Bulk Media Conversion activity to print the report.

Reserved File Codes

Most file codes beginning with the character "S" have special meaning to the Sort. (Such file codes include S1,S2,...,SA,SB,...,SZ.) In any COBOL program which utilizes the Sort, the file codes specified in all SELECT sentences should therefore definitely begin with a character other than "S".

APPENDIX-K

OPTIMIZING THE COMPILATION PROCESS

FOR GE-625/635 COBOL SOURCE PROGRAMS

GENERAL DESCRIPTION

An optional feature has been implemented in the COBOL compiler which can reduce processing time by as much as 50% when compiling programs containing source errors.

The entire compilation process can be logically divided into two broad phases of activity.

1. In Phase-1, the source program is thoroughly analyzed to ensure lexical, syntactic and semantic accuracy. It is simultaneously being translated into an intermediate language suitable for further processing.
2. In Phase-2, the intermediate language created in Phase-1 is converted to the appropriate machine language code and assembled into an executable object program.

Since all of the compiler diagnostic functions, error message preparation and cross-referenced source listing are completed in Phase-1, it is normally wasteful to continue the compilation beyond Phase-1 when source errors are detected. In most instances the object program produced from an erroneous source program is quite useless.

This option is selected by means of a single entry in the SPECIAL-NAMES paragraph. (See Pages VIII-4 and VIII-4.3.)

FUNCTION

Substantially reduce compilation time by eliminating redundant processing when source program errors exist.

METHOD OF IMPLEMENTATION

Normal Procedure

The standard procedure adopted, when this option is not requested, is to continue compilation through Phase-2 and produce an object program, irrespective of any source errors encountered.

SPECIAL-NAMES Paragraph Entry

To utilize the optional feature, an entry must be made in the Special-Names paragraph as shown below:

OPTION 8:

SPECIAL-NAMES .

COMPILE PHASE1 ONLY WITH SOURCE ERRORS.

Usage Notes:

1. When this statement is used in the source program, the extent of the compilation is determined by the presence or absence of source errors.
2. The following action is taken by the compiler:
 - If a source program error is detected, the compilation will be terminated at the end of Phase-1 and no assembly listing or object program will be produced.
 - If no source errors exist, then a normal compilation through assembly and executable object program will occur.
3. Only true error conditions will trigger the option. Warning and Efficiency messages are not significant.
4. A saving of up to 50% of compile time can be achieved when this feature is used for the initial compilation of new programs. However, it may be used with all programs as a standard procedure since it has no effect on compiler output if the source program is error-free.

5. No USEFUL output is suppressed when this option is selected. The source listing with cross-references and messages is always produced.
6. This feature is unique to GE-625/635 COBOL. The statement would therefore have to be removed before being compiled on another computer line.

APPENDIX L

USE OF WRITE VERB FOR LISTING FILES

An implicit report code is generated when the WRITE ... ADVANCING option is used for a listing file. The report code selected is the second character of the two character file code assigned to the file.

The use of a simple WRITE (no ADVANCING option), causes a report code of zero to be created. If a program has a mixture of simple WRITE statements and WRITE ... ADVANCING statements, there may be two different report codes used. This results in a segregation of listing output. This segregation can be avoided by assigning a file code which has a second character of zero.

EXAMPLE

```
SELECT ... ASSIGN TO ZA FOR LISTING
```

- S1. WRITE record-name-1. (Report code is 0)
- S2. WRITE record-name-2 ... ADVANCING ... (Report code is A)

Assuming S1 is executed first, all output with a report code of zero would be printed before the output with a report code of A. If S2 were executed first, all output with a report code of A would be printed before the output with a report code of zero.

If the listing file was assigned a file code of Z0, there would be no segregation of output.

Note that if more than one listing file is open, file codes unique in the second character must be assigned to the files to avoid intermingled output. In addition, only the WRITE ... ADVANCING option should be used for all but possibly one of the listing files. The exception would be for the one file whose second character of the file code is ZERO. Simple WRITES could be issued to this file.

APPENDIX M

REPORT WRITER PRE- AND POST-SLEW CALCULATION

The purpose of this appendix is to present, with examples, the algorithm used by the COBOL Report Writer in making Pre- and Post-Slew calculations.

Pre-Slew

The basic algorithm is:

LINE PLUS N slews N-1 lines.

Therefore:

LINE PLUS 0 → pre-slew 0 lines.
LINE PLUS 1 → pre-slew 0 lines.
LINE PLUS 2 → pre-slew 1 line.
LINE PLUS 3 → pre-slew 2 lines.

Exception:

LINE PLUS 0 → pre-slew 0 lines NOT N-1 lines.

Post-Slew

The basic algorithm is:

NEXT GROUP PLUS M slews M lines.

Therefore:

NEXT GROUP PLUS 0 → post-slew 0 lines.
NEXT GROUP PLUS 1 → post-slew 1 line.
NEXT GROUP PLUS 2 → post-slew 2 lines.
NEXT GROUP PLUS 3 → post-slew 3 lines.

Exception:

IF NEXT GROUP NOT SPECIFIED → automatic post-slew 1 line.

Combinations of Pre- And Post-Slew

LINE PLUS 0	————▶	pre-slew 0, post-slew 1
LINE PLUS 1	————▶	pre-slew 0, post-slew 1
LINE PLUS 2	————▶	pre-slew 1, post-slew 1
LINE PLUS 0		
NEXT GROUP PLUS 0	————▶	pre-slew 0, post-slew 0
LINE PLUS 1		
NEXT GROUP PLUS 0	————▶	pre-slew 0, post-slew 0
LINE PLUS 2		
NEXT GROUP PLUS 0	————▶	pre-slew 1, post-slew 0
LINE PLUS 1		
NEXT GROUP PLUS 1	————▶	pre-slew 0, post-slew 1
LINE PLUS 2		
NEXT GROUP PLUS 1	————▶	pre-slew 1, post-slew 1
LINE PLUS 1		
NEXT GROUP PLUS 2	————▶	pre-slew 0, post-slew 2
LINE PLUS 2		
NEXT GROUP PLUS 2	————▶	pre-slew 1, post-slew 2

Thus:

1. For normal single spacing of a report, a LINE PLUS 1 designation is most often used. It actually gives a pre-slew of 0; however, the implicit post-slew of 1 yields the desired result.
2. For normal double spacing of a report, a LINE PLUS 2 gives a pre-slew of 1 which when added to implicit post-slew of 1 from the previous line results in a double spaced report.
3. If line over-print is desired in the event a detail line must be split into 2 TYPE DETAIL Report Groups, this may easily be accomplished by using a LINE PLUS 0 NEXT GROUP PLUS 0 on the first TYPE DETAIL to be generated. The remainder of the detail line generated by the second TYPE DETAIL would specify LINE PLUS 0. This would result in both TYPE DETAIL LINES being printed on the same line.

APPENDIX N

REPORT WRITER TABLE CAPACITY

INTRODUCTION

The purpose of this appendix is to provide the COBOL Report Writer user with information concerning Report Table capacity. The Report Table is fixed length within the COBOL Compiler and has no overflow capability.

Since large report group descriptions, normally TYPE DETAIL or CONTROL FOOTING, occasionally overflow the Report Table capacity, this document is intended as an aid to the user in structuring his report group descriptions to fit within the Report Table capacity.

Report Group Entries

A report group entry is built for the 01 statement which contains the TYPE clause. Only one such entry appears per report group. The entry varies from a minimum of 8 words to a maximum of 11 words.

A basic 01 report group entry containing only a TYPE statement is built as an 8 word entry. For example:

```
01 DET-L TYPE DE.
```

This statement is built as an 8 word report group entry.

If a LINE or NEXT GROUP designation or a combination of both appear in the 01 report group statement, two additional words are required in the entry being built. For example:

```
01 DET-X TYPE DE LINE PLUS 1. or  
01 DET-Y TYPE DE NEXT GROUP PLUS 2. or  
01 DET-Z TYPE DE LINE PLUS 1 NEXT GROUP PLUS 2.
```

All of these statements build a 10 word entry.

If the report group is a CONTROL FOOTING, one additional word is required in the entry being built which is used as a TOTAL SUM WORD COUNT. For example:

```
01 TYPE CF data-name LINE PLUS 1 NEXT GROUP NEXT PAGE.
```

This statement is built as an 11 word entry.

Group Entries

In structuring a report group, the programmer may specify intervening group levels. Each group statement at 02 level or below requires a group entry which is nearly identical to the report group entry described above. It may vary in size from 8 words minimum to 10 words maximum. For example, consider the following structure:

```
01 CTL-X TYPE CF CNTRL NEXT GROUP NEXT PAGE.  
    02 LINE PLUS 1.  
        03 COLUMN 1 PICTURE Z(6) SUM data-name.  
    02 LINE PLUS 2.  
        03 . . .
```

The 01 TYPE statement would build an entry 11 words in length since it has a NEXT GROUP clause and is a TYPE CONTROL FOOTING. The 02 group statement builds a 10 word entry inheriting most of its data from the report group entry and adding the word of data to describe the line clause.

Source Entries

A source entry is built for each elementary statement containing a SOURCE IS clause. These entries can vary greatly in size, from a minimum entry of 24 words to a maximum entry of 71 words.

The most basic SOURCE statement is found in a detail report group. It does not contain a COLUMN clause and is therefore not printed. It is required to define a sum counter for a CONTROL FOOTING report group. For example:

```
02 SOURCE IS data-name.
```

This statement is built as a 24 word entry.

When the SOURCE statement contains subscripted items, the number of words required increases rapidly. One data-name subscript adds 12 additional words to the entry built. For example:

```
02 SOURCE IS data-name (data-name). 36 words
```

A single literal subscript requires 9 additional words. For example:

```
02 SOURCE IS data-name (literal). 33 words
```

Each subsequent subscript in the same statement adds 11 words in the case a data-name or 8 words in the case of a literal. For example:

02 SOURCE IS data-name (dn, dn-1).	47 words
02 SOURCE IS data-name (lit, lit-1).	41 words
02 SOURCE IS data-name (dn, lit).	44 words
02 SOURCE IS data-name (dn, dn-1, dn-2).	58 words
02 SOURCE IS data-name (lit, lit-1, lit-2).	49 words
02 SOURCE IS data-name (dn, dn-1, lit).	55 words
02 SOURCE IS data-name (dn, lit, lit-1).	52 words

When a COLUMN clause is added, it designates a receiving field must be provided in the source entry being built. The entry varies from 10 to 13 words in length depending on whether or not editing is required in the receiving field. For example:

```
02 COLUMN 1 PICTURE 9(6) SOURCE IS data-name.
```

This would not require editing so the total entry built would be 24 words plus 10 or 34 words. If the statement were:

```
02 COLUMN 1 PICTURE ZZZ,ZZ9.99 SOURCE IS data-name.
```

Editing is required, thus the entry would require 24 words plus 13 or 37 words.

The largest possible entry would therefore be of the type:

```
02 COLUMN 1 PICTURE ZZ,999 SOURCE IS data-name (dn, dn-1, dn-2).
```

This would require 71 words.

Sum Entries

A sum entry is built for each elementary statement containing a SUM clause. Sum entries can vary in size from a minimum of 34 words to a maximum of 37 words.

The basic SUM statement is found in the TYPE CONTROL FOOTING report group and usually is of the format:

```
02 CTL-X COLUMN 1 PICTURE ZZZ,ZZ9.99 SUM data-name.
```

or

```
02 COLUMN 1 PICTURE ZZZ,ZZ9.99 SUM data-name.
```

Either of the above statements will be built as an entry 37 words in length.

It is interesting to note that the addition of subscripts, either data-name or literal does not increase the size of the entry to be built. Thus, the statement:

```
02 COLUMN 1 PICTURE Z(6).99. SUM data-name (dn, dn-1, dn-2).
```

requires the same number of words as

```
02 COLUMN 1 PICTURE Z(6).99 SUM data-name.
```

The variance between 34 and 37 word entries is due to receiving field editing requirements and is described under COLUMN statement of SOURCE entry description.

Value Entries

A VALUE entry is built for each elementary item which contains a VALUE clause. Since the VALUE clause expresses a literal which can range from a single character to 132 characters in length, it follows that the entry built for a VALUE clause varies in the same proportion. For example:

```
02 COLUMN 1 PICTURE X VALUE "-".
```

represents a minimum entry and requires 26 words. On the other hand, the statement:

```
02 COLUMN 1 SIZE 132 VALUE "132 character literal---".
```

represents a maximum entry and requires a 48 word entry.

The following table indicates the entry size for the various literal sizes.

LITERAL SIZE (CHARACTERS)	VALUE ENTRY (WORDS)
1-2	26
3-8	27
9-14	28
15-20	29
21-26	30
27-32	31
33-38	32
39-44	33
45-50	34
51-56	35
57-62	36
63-68	37
69-74	38
75-80	39
81-86	40
87-92	41
93-98	42
99-104	43
105-110	44
111-116	45
117-122	46
123-128	47
129-132	48

EXCEPTIONS

Two possible exceptions may change the maximum entry sizes as indicated in this discussion. Both are rather remote in usage but bear clarification.

1. If an elementary SOURCE, SUM or VALUE statement contains a LINE clause as part of its description, the entry built is increased by two words. For example:

```
02 LINE PIJS 1 COLUMN 1 PICTURE X(6) SOURCE data-name.
```

This statement builds a source entry of 36 words as compared to 34 words if the LINE clause does not appear at the elementary level.

Sum and value entries reflect the same two word increase in size.

2. In the receiving field description designated by the COLUMN and PICTURE clauses, if editing is required and the receiving field is over 38 characters in length, one additional word is required. If over 76 characters in length, two additional words are required.

CALCULATION OF REPORT GROUP SIZE

The length of the variable entry portion of the Report Table in most cases is 1650 words; however, in certain instances, it can be reduced to a minimum of 1260 words.

This reduction occurs only when the following conditions exist:

- a. The report description (RD) contains a large number of report groups.
- b. The larger report groups appear near or at the end of the report description.

A rule of thumb to follow in report description organization is to place the larger report groups at the beginning of the report description. In this way, the maximum variable Report Table size of 1650 words will nearly always be available.

The following example shows the calculation of report group size with table capacity reduced to 1260 words because of one of the conditions previously mentioned.

```

.
.
.
01 DETL-X TYPE DE.                                     8
  02 LINE PLUS 2.                                     10
    03 COLUMN 1 PICTURE Z(6).99 SOURCE data-name (dn, dn-1, dn-2). 71
    03 COLUMN 15 PICTURE Z(6).99 SOURCE data-name-1 (dn, dn-1, dn-2). 71
    03 COLUMN 30 PICTURE Z(6).99 SOURCE data-name-2 (dn, dn-1, dn-2). 71
    03 COLUMN 45 PICTURE Z(6).99 SOURCE data-name-3 (dn, dn-1, dn-2). 71
    03 COLUMN 60 PICTURE Z(6).99 SOURCE data-name-4 (dn, dn-1, dn-2). 71

  02 LINE PLUS 3.                                     10
    03 COLUMN 1 PICTURE Z(6).99 SOURCE data-name-5 (dn, dn-1, dn-2). 71
    03 COLUMN 15 PICTURE Z(6).99 SOURCE data-name-6 (dn, dn-1, dn-2). 71
    03 COLUMN 30 PICTURE Z(6).99 SOURCE data-name-7 (dn, dn-1, dn-2). 71
    03 COLUMN 45 PICTURE Z(6).99 SOURCE data-name-8 (dn, dn-1, dn-2). 71
    03 COLUMN 60 PICTURE Z(6).99 SOURCE data-name-9 (dn, dn-1, dn-2). 71

  02 LINE PLUS 4.                                     10
    03 COLUMN 1 PICTURE Z(6).99 SOURCE data-name-10 (dn, dn-1, dn-2). 71
    03 COLUMN 15 PICTURE Z(6).99 SOURCE data-name-11 (dn, dn-1, dn-2). 71
    03 COLUMN 30 PICTURE Z(6).99 SOURCE data-name-12 (dn, dn-1, dn-2). 71
    03 COLUMN 45 PICTURE Z(6).99 SOURCE data-name-13 (dn, dn-1, dn-2). 71
    03 COLUMN 60 PICTURE Z(6).99 SOURCE data-name-14 (dn, dn-1, dn-2). 71

  02 LINE PLUS 5.                                     10
    03 COLUMN 1 PICTURE Z(6).99 SOURCE data-name-15 (dn, dn-1, dn-2). 71
    → 03 COLUMN 15 PICTURE Z(6).99 SOURCE data-name-16 (dn, dn-1, dn-2). 71
    03 COLUMN 30 PICTURE Z(6).99 SOURCE data-name-17 (dn, dn-1, dn-2). 71

```

At this point report group capacity is exceeded and the detail report group must be subdivided into two detail report groups.

This can be accomplished by dividing the report group into two report groups of the same type. In the case of the TYPE DETAIL report group, it requires insertion of an "01 DATA-NAME TYPE DE." statement and an additional GENERATE statement in the Procedure Division. If the overflowed group is a CONTROL FOOTING, the subdivision is a bit more complex. A dummy CONTROL DATA-NAME with the same PICTURE and USAGE as the original must be defined in WORKING STORAGE. Immediately after each READ of the pertinent Input File, the field which comprises the original CONTROL DATA-NAME must be moved to the dummy CONTROL DATA-NAME in WORKING STORAGE. The dummy CONTROL DATA-NAME becomes the CONTROL DATA-NAME for the new 01 TYPE CF report group and is also inserted in the CONTROLS ARE clause of the corresponding RD entry.

For example, to subdivide the following TYPE CF report group:

```
RD REPORT X CONTROLS ARE FINAL, DN1, DN2.  
01 TYPE CF DN1.  
    02 . . .
```

A dummy CONTROL DATA-NAME with the same PICTURE and USAGE as the original must be defined in WORKING STORAGE:

```
77 DN1A PICTURE 9(6).
```

After each READ of the pertinent Input File containing CONTROL DATA-NAME DN1, the new value of DN1 must be moved to DN1A before the corresponding GENERAGE statement for the REPORT being produced:

```
READ INPUT-FILE AT END GO TO ---.  
MOVE DN1 TO DN1A.  
GENERATE DETAIL-1.
```

The report group subdivided would be:

```
RD REPORT X CONTROLS ARE FINAL, DN1A, DN1, DN2.  
01 TYPE CF DN1.  
    02 . . .  
    .  
    .  
    .  
    02 . . .  
01 TYPE CF DN1A.  
    02 . . .
```

The control break for DN1 and DN1A will occur at the same time. The CONTROL FOOTING report groups are presented from minor to major; therefore, the report group with DN1 will be produced before the report group with DN1A. The order may be adjusted as required by the program needs.

APPENDIX O

COBOL EXAMPLES

This appendix is intended to serve as a guide for some common usages of GE-625/635 COBOL. Many subjects are not included here if in particular they have been the subject of another appendix. The reader is referred to the following appendixes:

- A. Computational Item Formats
- B. {
- C. { File Formats and Processing
- D. {
- E. Efficiency Techniques
- H. {
- J. { Use of COBOL SORT

This appendix includes sample verb results and sample programs.

Sample Verb Results

The effect on sample data is illustrated here for two common verbs. The first example applies to Numeric Edited MOVE results and is given in tabular form for various Pictures. The second example shows the effect on various data items when a Conditional Relational (simple IF) Test is applied.

Numeric Edited MOVE Examples

Sending Item		Receiving Item	
PICTURE	VALUE	PICTURE	Resulting Value
9(5)	45678	\$ZZ,ZZ9.99	\$45,678.00
9(3)V99	456.78	\$ZZ,ZZ9.99	\$ 456.78
9(3)V99	000.67	\$ZZ,ZZ9.99	\$ 0.67
9(3)V99	000.04	\$ZZ,ZZZ.99	\$.04
9(5)	00000	\$ZZ,ZZZ.ZZ	
V9(5)	.12345	\$ZZ,ZZ9.99	\$ 0.12
9(5)	12345	\$**,**9.99	\$12,345.00
9(5)	67890	\$\$\$,\$\$9.99	\$67,890.00
9(3)V99	678.90	\$\$\$,\$\$9.99	\$678.90
9(5)	00000	\$\$\$,\$\$9.99	\$0.00
V9(5)	.67890	\$\$\$,\$\$9.99	\$0.67
S9(5)V	-56789.	-ZZZZ9.99	-56789.00
S9(5)	+56789	-ZZZZ9.99	56789.00
S9(5)	-56789	+ZZZZ9.99	-56789.00
S9(5)	+56789	+ZZZZ9.99	+56789.00
S99V9(3)	-56.789	-----9.99	-56.78
S9(5)	-00567	ZZZZZ.99-	567.00-
S9(5)	-56789	\$\$\$\$\$.99CR	\$56789.00CR
S9(5)	+56789	\$\$\$\$\$.99CR	\$56789.00

Conditional Relation Test Examples

Item A		Item B		Relation
PICTURE	VALUE	PICTURE	VALUE	

Numeric Items

9(6)V9	1.2	9V9	6.5	A < B
9(6)V99	1.33	9V999	1.330	A = B
S9V9999	-0.025	S99V99999	-00.064	A > B
99	00	V999	.000	A = B

Nonnumeric Items of Equal Size

99.9	01.2	99.9	12.4	A < B
+999.9	+003.1	+999.9	+003.0	A > B
X(4)	ABCD	X(4)	TAXY	A < B
X(4)	0123	X(4)	+001	A < B
X(4)	(24)	X(4))01(A < B
X(4)	A1B2	X(4)	A1B2	A = B

Nonnumeric Items of Unequal Size

\$999.99	\$123.45	\$99.9	\$12.6	A < B
X(5)	ABCDE	X(3)	ABC	A > B
X(5)	ABCΔΔ	X(3)	ABC	A = B
X(5)	ABCDE	X(3)	FCE	A < B
X(2)	ΔΔ	X(4)	0100	A > B

Standard Collating Sequence ←

Sample Programs

Two sample programs are shown here for general illustrative purposes. The first program (REPRT) is intended to be a guide to the use of the Report Writer in COBOL. It produces one report with three actual control breaks, which is shown as a two-page sample output. By referring to the various report groups in the program, the effect on the output report may be readily seen (including the various line slewing controls). Notice also that a dummy control footing is used to force the final control break to the next page. The Procedure Division illustrates the minimum necessary statements to produce the same report from input data coming from cards.

000010	IDENTIFICATION DIVISION.
000020	PROGRAM-ID. REPRT.
000030	REMARKS.
000040	THIS EXAMPLE PRODUCES AN OUTPUT REPORT VIA REPORT WRITER. THE REPORT CONSISTS OF ONE
000050	BASIC DETAIL LINE WITH 3 CONTROL BREAKS--ONE
000060	OF WHICH IS A FINAL CONTROL BREAK.
000070	ENVIRONMENT DIVISION.
000080	CONFIGURATION SECTION.
000090	SPECIAL-NAMES.
000100	"A" IS CODE-1,
000110	INPUT-OUTPUT SECTION.
000120	FILE-CONTROL.
000130	SELECT INPUT-FILE ASSIGN TO AA FOR CARDS.
000140	SELECT OUTPUT-FILE-1 ASSIGN TO BB FOR LISTING.
000150	I-O-CONTROL.
000160	DATA DIVISION.
000170	FILE SECTION.
000180	FD INPUT-FILE
000190	LABEL RECORDS ARE STANDARD
000200	DATA RECORD IS CARD-INPUT.
000210	01 CARD-INPUT.
000220	02 C-IN.
000230	04 CONTROL-FIELD.
000240	08 CONTROL-1 PICTURE 9.
000250	08 CONTROL-2 PICTURE 9.
000260	08 CONTROL-3 PICTURE 9.
000270	04 FILLER PICTURE X(6).
000280	04 ITEM-VOLUMN PICTURE 9(6).
000290	04 COST PICTURE 9(10)V99.
000300	04 TITLES PICTURE X(50).
000310	FD OUTPUT-FILE-1
000320	LABEL RECORDS ARE STANDARD
000330	REPORT IS REPORT-1.
000340	WORKING-STORAGE SECTION.
000350	01 CARD-IN-WS.
000360	04 CONTROL-FIELD PIC 999.
000370	04 FILLER PIC X(6).
000380	04 ITEM-VOLUMN PIC 9(6).
000390	04 COST PIC 9(10)V99.
000400	04 TITLES PIC X(50).
000410	REPORT SECTION.
000420	RD REPORT-1
000430	WITH CODE CODE-1, PAGE LIMIT IS 55 LINES
000440	CONTROLS ARE FINAL, CONTROL-3, CONTROL-1, CONTROL-2.
000450	01 REPORT-HEADING-1, TYPE REPORT HEADING.
000460	03 LINE 02.
000470	05 COLUMN 56 PICTURE X(20) VALUE " REPORT HEADING " .
000480	01 PAGE-HEADING, TYPE PAGE HEADING.
000490	03 LINE 03.
000500	05 COLUMN 56 PICTURE X(20) VALUE " PAGE HEADING " .
000510	05 COLUMN 117 PICTURE X(5) VALUE "PAGE " .
000520	05 COLUMN 122 PICTURE Z9 SOURCE IS PAGE-COUNTER.
000530	01 DETAIL-LINE-1 TYPE DETAIL LINE PLUS 1 NEXT GROUP PLUS 1.

000550	02	SOURCE SELECTED CARD-INPUT.
000560*		NOTE--NO QUALIFICATION IS NECESSARY ON SOURCE ITEMS
000570	03	COLUMN 7 PICTURE XXX SOURCE IS CONTROL-FIELD.
000580	03	COLUMN 54 PICTURE X(50) SOURCE IS TITLES.
000590	03	COLUMN 105 PICTURE ZZZZ9 SOURCE IS ITEM-VOLUMN.
000600	03	COLUMN 115 PICTURE \$*,***,***,**9.99 SOURCE IS COST.
000610	01	CONTROL-HEADING-1, TYPE CONTROL HEADING CONTROL-1.
000620	03	LINE PLUS 01.
000630	05	COLUMN 2 PICTURE X(20) VALUE " CONTROL-1 HEADING " .
000640	05	COLUMN 50 PICTURE X(21) VALUE "--ITEM DESCRIPTION--".
000650	01	CONTROL-FOOTING-1, TYPE CONTROL FOOTING CONTROL-1.
000660	02	LINE PLUS 1, NEXT GROUP PLUS 2.
000670	03	COLUMN 25 PICTURE X(17) VALUE "VOLUME SUB-TOTAL=".
000680	03	COLUMN 42 PICTURE ZZZZZ9 SUM ITEM-VOLUMN OF C-IN.
000690	02	LINE PLUS 0, NEXT GROUP PLUS 2.
000700	03	COLUMN 2 PICTURE X(27) VALUE " END OF CONTROL-1 FOOTING".
000710	01	CONTROL-HEADING-2, TYPE CONTROL HEADING CONTROL-2.
000720	03	LINE PLUS 01.
000730	05	COLUMN 2 PICTURE X(20) VALUE " CONTROL-2 HEADING " .
000740	01	CONTROL-FOOTING-2, TYPE CONTROL FOOTING CONTROL-2.
000750	02	LINE PLUS 2.
000760	03	COLUMN 25 PICTURE X(15) VALUE "COST SUB-TOTAL=".
000770	03	COLUMN 40 PICTURE \$*,***,***,**9.99 SUM COST OF C-IN.
000780	02	LINE PLUS 2, NEXT GROUP PLUS 2.
000790	03	COLUMN 2 PICTURE X(27) VALUE " END OF CONTROL-2 FOOTING".
000800*		NOTE--THIS IS A DUMMY CF TO FORCE FINAL CONTROL BREAK TO
000810*		NEXT PAGE
000820	01	CONTROL-FOOTING-3 TYPE CONTROL FOOTING CONTROL-3,
000830		LINE PLUS 1, NEXT GROUP NEXT PAGE.
000840	02	COLUMN 1 PICTURE X VALUE SPACE.
000850	01	CONTROL-HEADING-FINAL, TYPE CONTROL HEADING FINAL.
000860	03	LINE PLUS 2.
000870	05	COLUMN 2 PICTURE X(20) VALUE " FINAL HEADING " .
000880	05	COLUMN 107 PICTURE X(22) VALUE "VOL. COST".
000890	01	CONTROL-FOOTING-FINAL TYPE CONTROL FOOTING FINAL.
000900	02	LINE PLUS 3.
000910	03	COLUMN 93 PICTURE X(10) VALUE "TOTAL VOL." .
000920	03	COLUMN 115 PICTURE X(10) VALUE "TOTAL COST" .
000930	02	LINE PLUS 2.
000940	03	COLUMN 93 PICTURE ZZZZZ9 SUM ITEM-VOLUMN OF C-IN.
000950	03	COLUMN 115 PICTURE \$*,***,***,**9.99 SUM COST OF C-IN.
000960	02	LINE PLUS 2, NEXT GROUP PLUS 3.
000970	03	COLUMN 2 PICTURE X(27) VALUE " END OF FINAL FOOTING " .
000980	01	PAGE-FOOTING, TYPE PAGE FOOTING.
000990	03	LINE PLUS 2.
001000	05	COLUMN 56 PICTURE X(20) VALUE " PAGE FOOTING " .
001010	01	REPORT-FOOTING, TYPE REPORT FOOTING.
001020	03	LINE 54.
001030	05	COLUMN 56 PICTURE X(20) VALUE " REPORT FOOTING " .
001040		PROCEDURE DIVISION.
001050	AA10.	OPEN INPUT INPUT-FILE, OUTPUT OUTPUT-FILE-1.
001070	BB10.	INITIATE REPORT-1.
001080	CC10.	READ INPUT-FILE AT END GO TO EE10.

001090	DD10.	GENERATE	DETAIL-LINE-1.
001100	DD99.	GO TO	CC10.
001110	EE10.	TERMINATE	REPORT-1.
001130	FF10.	CLOSE	INPUT-FILE, OUTPUT-FILE-1.
001150	ZZ99.	STOP	RUN.

GE-600 SERIES

8-0

COBOL

REPORT HEADING		PAGE 1	
PAGE HEADING			
FINAL HEADING	--ITEM DESCRIPTION--	VOL.	COST
CONTROL-1 HEADING			
CONTROL-2 HEADING			
00	SMALL MOTORS - # M12J50	100	\$*****39,50
00	FANS - #F160	25968	\$*****6,98
COST SUB-TOTAL=\$*****46,48			
END OF CONTROL-2 FOOTING			
VOLUME SUB-TOTAL= 26068			
END OF CONTROL-1 FOOTING			
CONTROL-1 HEADING	--ITEM DESCRIPTION--		
CONTROL-2 HEADING			
11	LARGE BOLTS - # BXJ9	100000	\$*****0,39
11	MACHINE SCREWS - # SX15	59000	\$*****0,15
COST SUB-TOTAL=\$*****0,54			
END OF CONTROL-2 FOOTING			
VOLUME SUB-TOTAL= 159000			
END OF CONTROL-1 FOOTING			
CONTROL-1 HEADING	--ITEM DESCRIPTION--		
CONTROL-2 HEADING			
22	LARGE MOTORS - # ML30	20	\$*****3,000,00
22	JET ASSEMBLY - # JL65	5	\$*****650,000,00
COST SUB-TOTAL=\$*****653,000,00			
END OF CONTROL-2 FOOTING			
CONTROL-2 HEADING			
23	PIPER CUB PLANE - # PC100	1	\$*****10,000,00
23	JET PLANE (4 MOTOR) - # LJ15	1	\$****1,500,000,00
COST SUB-TOTAL=\$****1,510,000,00			
END OF CONTROL-2 FOOTING			
VOLUME SUB-TOTAL= 27			
END OF CONTROL-1 FOOTING			
PAGE FOOTING			

PAGE HEADING

PAGE 2

TOTAL VOL,
185095

TOTAL COST
S****2,163,047,02

END OF FINAL FOOTING

PAGE FOOTING

REPORT FOOTING

The second sample program (UPDATE) illustrates a common type of business application, namely, updating of a master file with insertions, deletions, and modifications of file records. The problem definition is oversimplified for the purpose of an example and no claims are made as to its error-free execution within this context. However, liberal use is made of debug statements to follow the processing of input data and may be helpful as a guide to their use for debugging programs. This program may also be useful as a reference for the more common usages of I/O, PERFORM, MOVE, and IF statements.

```

000000 IDENTIFICATION DIVISION.
000010 PROGRAM-ID. UPDATE.
000020 REMARKS.          THIS EXAMPLE UPDATES A MASTER FILE AND
000030                      PRODUCES A COMPLETE LISTING OPTIONALLY BY
000040                      MEANS OF A CONTROL CARD. CHANGES TO A MASTER
000050                      FILE ARE CARD INPUT ALONG WITH CERTAIN
000060                      CONTROL CARDS WHICH ALLOW:
000070
000080                      1. INSERTIONS AFTER GIVEN SEQUENCE NUMBER
000090                      2. MODIFICATIONS OF DATA OR SEQUENCE NUMBERS ONLY
000100                      3. DELETES DATA FROM ONE SEQUENCE NUMBER TO ANOTHER AND
000110                      INSERTS FOLLOWING DATA CARDS.
000120
000130 CONTROL CARDS ARE ASSUMED TO BE SORTED BY SEQ.# ON INPUT.
000140 SEQUENCE NUMBERS NOT MODIFIED ARE LEFT UNCHANGED.
000150
000160 CONTROL-CARD FORMATS: COL1      7          19
000170                      LIS 999999 (LISTING REQUEST CARD WHERE
000180                      999999 DEFINES THE SEQ.# #
000190                      INSERTION INCREMENT)
000200
000210                      *IN 999999 (INSERTS FOLLOWING CARDS
000220                      AFTER SEQ.# 999999)
000230
000240                      *MF NNNNNN TO MMMMM
000250                      (MODIFIES CARD WITH SEQ.# NNNNNN
000260                      WITH FOLLOWING CARD-IF PRESENT-
000270                      AND CHANGES SEQ.# TO MMMMM)
000280
000290                      *DE NNNNNN THRU MMMMM
000300
000310                      (DELETES CARDS FROM SEQ.# NNNNNN
000320                      THRU MMMMM AND REPLACES WITH ANY
000330                      FOLLOWING CARDS)
000330 ENVIRONMENT DIVISION.
000340 CONFIGURATION SECTION.
000350 SPECIAL-NAMES.
000360          SYSOUT IS TYP, GETIME IS GTME, GELAPS IS GLAPS,
000370          COMPILE PHASE1 ONLY WITH ERRORS,
000380          PROCESS ALL DEBUG STATEMENTS.
000390 *NOTE--ABOVE CLAUSE MAY BE VARIED OR REMOVED TO FACILITATE DEBUG
000400 INPUT-OUTPUT SECTION.
000410 FILE-CONTROL.
000420          SELECT CARD-FILE  ASSIGN TO CD  FOR CARDS.
000430          SELECT LIST-FILE  ASSIGN TO LM  FOR LISTING.
000440          SELECT OLD-MASTER ASSIGN TO OM.
000450          SELECT NEW-MASTER ASSIGN TO NM.
000460 I-O-CONTROL.
000470          APPLY STANDARD ON OLD-MASTER, NEW-MASTER.
000480 DATA DIVISION.
000490 FILE SECTION.
000500 FD NEW-MASTER LABEL RECORDS STANDARD, DATA RECORD IS MAS-OUT.

```

```

000510 01 MAS-OUT.
000520 02 MLINE PICTURE X(74).
000530 02 MSEQ PICTURE 9(6).
000540 FD OLD-MASTER LABEL RECORDS ARE STANDARD, DATA RECORD IS MAS-IN.

000550 01 MAS-IN.
000560 02 ILINE PICTURE X(74).
000570 02 ISEQ PICTURE 9(6).
000580 FD CARD-FILE LABEL RECORDS STANDARD, DATA RECORD IS CARD-REC.
000590 01 CARD-REC.

000600 02 C-GR.
000610 03 CLINE PICTURE X(74).
000620 03 CSEQ PICTURE 9(6).
000630 02 CR REDEFINES C-GR.
000640 03 CARD-FLD PICTURE X(3) OCCURS 26 TIMES.
000650 03 FILLER PICTURE XX.
000660 02 CS REDEFINES CR.
000670 03 CON-FLD PICTURE 9(6) OCCURS 13 TIMES.
000680 03 FILLER PICTURE XX.
000690 FD LIST-FILE LABEL RECORDS ARE STANDARD, DATA RECORDS ARE
000700 LIST-REC, DATE-REC.
000710 01 LIST-REC.
000720 02 L-FLD PICTURE X(74).
000730 02 LSEQ PICTURE 9(6).
000740 02 FILL PICTURE X(4).

000750 01 DATE-REC.
000760 02 INDENT PICTURE X(28).
000770 02 DATE PICTURE X(8).
000780 * * * * *

000790 WORKING-STORAGE SECTION.
000800 77 OLD-MAS-IND PICTURE 9 VALUE 0.
000810 88 OLD-MAS-OPEN VALUE 1.
000820 88 OLD-MAS-CLOSED VALUE 0.
000830 77 LIST-ALL-IND PICTURE 9 VALUE 0.
000840 88 LIST-ON VALUE 1.
000850 88 LIST-OFF VALUE 0.
000860 77 WRITE-IND PICTURE 9 VALUE 0.
000870 88 WRIT-ON VALUE 1.
000880 88 WRIT-OFF VALUE 0.
000890 77 DISPLAY-ERR PICTURE X(6).
000900 77 CURRENT-SEQ PICTURE 9(6) VALUE 0.
000910 77 INCR PICTURE 9(6) VALUE 1.
000920 77 CX PICTURE 9(3) USAGE COMP-1.
000930 77 TIME-2 PICTURE 9(8) USAGE COMP-1.
000940 77 ELAPSED-TIM PICTURE ZZZ9.9999 .
000950 01 CON-REC.
000960 02 CRD-TYP.
000970 03 C-TYP PIC X(12) VALUE " *IN*MF*DE".
000980 02 TYP-ARR REDEFINES CRD-TYP.
000990 03 CON-TYP PICTURE XXX OCCURS 4 TIMES.

```

```

001000 01 INDATE.
001010      02 IDATE.
001020      03 MO      PICTURE 99.
001030      03 DA      PICTURE 99.

001040      03 YR      PICTURE 99.
001050      02 TIME-1  PICTURE 9(8) USAGE COMP-1.
001060 01 ODATE.
001070      02 ODATE.
001080      03 MCT      PICTURE 99.
001090      03 FILLER  PICTURE X VALUE "-".
001100      03 DAY      PICTURE 99.
001110      03 FILLER  PICTURE X VALUE "-".
001120      03 YRR     PICTURE 99.
001130 PROCEDURE DIVISION.
001140* * * * *

001150 START-UP SECTION. ACCEPT INDATE FROM GTME.
001160      MOVE MO TO MOT. MOVE DA TO DAY. MOVE YR TO YRR.
001170      MOVE TIME-1 TO TIME-2.
001180      OPEN INPUT CARD-FILE, OLD-MASTER,
001190              OUTPUT NEW-MASTER, LIST-FILE.
001200      MOVE 1 TO OLD-MAS-IND.
001210      MOVE SPACES TO INDENT. MOVE ODATE TO DATE.
001220      WRITE DATE-REC BEFORE ADVANCING 2 LINES.
001230      MOVE SPACES TO FILL.
001240      PERFORM READ-CD.
001250 NOTE - THE FOLLOWING IS A LEVEL 1 DEBUG STATEMENT.

0012601 DIS-1. DISPLAY CARD-REC UPON TYP.
001270      IF CARD-FLD (1) NOT = "LIS" GO TO CONTROL-CARD-TEST
001280              ELSE MOVE 1 TO LIST-ALL-IND.
001290      IF CON-FLD (2) NOT = " " MOVE CON-FLD (2) TO INCR.
001300 READ-CD. READ CARD-FILE AT END GO TO CLOSE-CARD.
001310 NOTE - THE FOLLOWING IS A LEVEL 2 DEBUG STATEMENT.
0013202 DIS-2. DISPLAY CARD-REC UPON TYP.
001330 CONTROL-CARD-TEST. MOVE 2 TO CX.
001340 BR-C. IF CARD-FLD (1) = CON-TYP (CX) GO TO BRANCH.

001350      ADD 1 TO CX IF CX < 5 GO TO BR-C.
001360      MOVE CARD-FLD (1) TO DISPLAY-ERR.
001370      DISPLAY "UNDEFINED CARD ", DISPLAY-ERR, UPON TYP.

001380      GO TO READ-CD.
001390 BRANCH. SUBTRACT 1 FROM CX.
001400      GO TO INS-RTN, MOD-RTN, DEL-RTN DEPENDING ON CX.
001410      STOP "DEAD END ".
001420* * * * *

001430 INS-RTN SECTION.
001440 II. IF OLD-MAS-OPEN GO TO READ-OLD.
001450 OLD-MAS-ERR. ADD INCR, CON-FLD (2) GIVING CURRENT-SEQ.
001460      MOVE 1 TO WRITE-IND GO TO INSERT-CARD.
001470 READ-OLD. IF WRIT-ON MOVE 0 TO WRITE-IND GO TO TEST-OLD.

```

```

001480      READ  OLD-MASTER  AT END MOVE 0 TO OLD-MAS-IND GO TO I1.
001490 TEST-OLD.  IF  CON-FLD (2) < ISEQ GO TO OLD-MAS-ERR.

001500      PERFORM PASS-AND-WRITE  GO TO READ-OLD.
001510 INSERT-CARD.  PERFORM READ-CD.
001520 NOTE - THE FOLLOWING IS A LEVEL 2 DEBUG STATEMENT.
0015302     DISPLAY CARD-REC UPON TYP.
001540      IF CARD-FLD (1) = CON-TYP (2) OR CON-TYP (3) OR
001550                CON-TYP (4) GO TO CONTROL-CARD-TEST.
001560      MOVE CURRENT-SEQ TO MSEQ, LSEQ.

001570 M-LIN.  MOVE CLINE TO MLINE, L-FLD.
001580 WRIT-2.  WRITE MAS-OUT, WRITE LIST-REC AFTER ADVANCING 1 LINE.
001590 END-IN.  ADD INCR TO CURRENT-SEQ GO TO INSERT-CARD.
001600 * * * * *
001610 MOD-RTN SECTION.
001620 M1.      IF  OLD-MAS-OPEN GO TO TEST-READ.
001630 MOD-ERR.  MOVE CON-FLD (2) TO DISPLAY-ERR.
001640      DISPLAY "BAD MODIFY CARD FOR ", DISPLAY-ERR UPON TYP.
001650      GO TO READ-CD.
001660 TEST-READ.  IF  WRIT-ON MOVE 0 TO WRITE-IND GO TO TEST-SEQ.
001670      READ  OLD-MASTER  AT END MOVE 0 TO OLD-MAS-IND GO TO M1.
001680 TEST-SEQ.  IF  CON-FLD (2) NOT = ISEQ PERFORM PASS-AND-WRITE

001690                                GO TO TEST-READ.

001695      MOVE CON-FLD (2) TO DISPLAY-ERR.
001700      IF  CON-FLD (4) NOT = "      " MOVE CON-FLD (4) TO
001710                MSEQ, LSEQ GO TO READ-NEXT-CARD
001720      ELSE MOVE ISEQ TO MSEQ, LSEQ.
001730 READ-NEXT-CARD.  READ  CARD-FILE  AT END PERFORM MOD-WARN
001740                GO TO CLOSE-CARD.
001750 NOTE - THE FOLLOWING IS A LEVEL 3 DEBUG STATEMENT.
0017603     DISPLAY CARD-REC UPON TYP.
001770      IF CARD-FLD (1) = CON-TYP (2) OR CON-TYP (3) OR
001780                CON-TYP (4) PERFORM MOD-WARN
001790                GO TO CONTROL-CARD-TEST.
001800      IF  CSEQ NOT = "      " MOVE CARD-REC TO MAS-OUT, LIST-REC

001805                                PERFORM WRIT-2
001810      ELSE PERFORM M-LIN THRU WRIT-2.
001820      GO TO READ-CD.
001830 * * * * *

001840 DEL-RTN SECTION.
001850 D1.      IF  OLD-MAS-OPEN GO TO READ-TEST.

001860 DEL-ERR.  MOVE CON-FLD (2) TO DISPLAY-ERR.

001870      DISPLAY "BAD DELETE CARD FOR ", DISPLAY-ERR, UPON TYP.
001880      MOVE 1 TO WRITE-IND GO TO READ-CD.
001890 READ-TEST.  IF  WRIT-ON MOVE 0 TO WRITE-IND GO TO SEQ-TEST.
001900      READ  OLD-MASTER  AT END MOVE 0 TO OLD-MAS-IND GO TO D1.

```

```

001910 SEQ-TEST. IF CON-FLD (2) > ISEQ PERFORM PASS-AND-WRITE
001920 GO TO READ-TEST.

001930 IF CON-FLD (2) < ISEQ GO TO DEL-ERR.

001940 PASS-NO-WRITE. READ OLD-MASTER AT END MOVE 0 TO OLD-MAS-IND
001950 GO TO OLD-MAS-ERR.
001960 IF CON-FLD (4) > ISEQ OR = ISEQ GO TO PASS-NO-WRITE

001970 ELSE MOVE 1 TO WRITE-IND MOVE CON-FLD (2) TO CURRENT-SEQ
001980 GO TO INSERT-CARD.
001990* * * * *
002000 PASS-AND-WRITE SECTION. MOVE MAS-IN TO MAS-OUT, LIST-REC.
002010 WRITE MAS-OUT IF LIST-OFF GO TO P1.
002020 WRITE LIST-REC AFTER ADVANCING 1 LINE.
002030 P1. MOVE ISEQ TO CURRENT-SEQ.
002040 NOTE - THIS IS EXIT.
002050* * * * *
002060 MOD-WARN SECTION.
002070 DISPLAY "NO MODIFY CARD TEXT FOUND FOR " DISPLAY-ERR,
002080 UPON TYP. MOVE ILINE TO MLINE, L-FLD.
002090 PERFORM WRIT-2.
002100 NOTE - THIS IS EXIT.
002110* * * * *
002120 CLOSE-CARD SECTION. CLOSE CARD-FILE IF OLD-MAS-CLOSED GO TO DONE.
002130 COPY-OLD. IF WRIT-ON MOVE 0 TO WRITE-IND GO TO C2.

002140 READ OLD-MASTER AT END MOVE 0 TO OLD-MAS-IND GO TO DONE.

002150 C2. PERFORM PASS-AND-WRITE GO TO COPY-OLD.
002160 DONE. CLOSE OLD-MASTER, NEW-MASTER, LIST-FILE.
002170 ACCEPT INDATE FROM GTME.

002180 COMPUTE ELAPSED-TIM ROUNDED = (TIME-1 - TIME-2) / 64000.
002190 DISPLAY "ELAP.CLOCK TIME(SEC)= ", ELAPSED-TIM, UPON TYP.
002200 ACCEPT TIME-2 FROM GLAPS.
002210 DIVIDE 64000 INTO TIME-2 GIVING ELAPSED-TIM.
002220 DISPLAY "ELAP.PROC.TIME(SEC) = ", ELAPSED-TIM, UPON TYP.
002230 STOP RUN.
002240 END OF PROGRAM.

```


APPENDIX P

COBOL SOURCE PROGRAM ORDER

<u>DIVISION</u>	<u>SECTION</u>	<u>Paragraph</u>	<u>Requ</u>
IDENTIFICATION DIVISION.			*
	any order	PROGRAM-ID. AUTHOR. INSTALLATION. DATE-WRITTEN. DATE-COMPILED. SECURITY. REMARKS.	*
ENVIRONMENT DIVISION.			*
	CONFIGURATION SECTION.		*
		SOURCE-COMPUTER. OBJECT-COMPUTER. SPECIAL-NAMES.	
	INPUT-OUTPUT SECTION.		
		FILE-CONTROL. I-O-CONTROL.	
DATA DIVISION.			
	FILE SECTION.		*
	any order	{ output files. input files.	
	WORKING-STORAGE SECTION.		
	CONSTANT SECTION.		
	REPORT SECTION.		
PROCEDURE DIVISION.			*
	ΔDECLARATIVES. (Use Sections)		
	ΔEND DECLARATIVES. (All Other Procedures)		
END PROGRAM.			

ΔIf used, both DECLARATIVES and END DECLARATIVES must be present.

APPENDIX Q

USE OF A LIBRARY FOR COPY

COBOL source cards may be inserted into a program from the user library file (.L) at compilation time by the use of either:

- a. the Data Division clause

COPY....FROM LIBRARY

- b. the Procedure Division - COPY verb

in a source program. By creation and use of a user library file lengthy repetitions of file, report, and record descriptions from the Data Division and paragraphs from the Procedure Division may be avoided in programs which use common data or procedures.

Source Library Format

When creating a library, there are some important source format restrictions that must be observed. They are:

1. All Data Division entries must be terminated with the PROCEDURE DIVISION card.
2. No reserved words must be used other than in their appropriate places, i.e.,

AUTHOR. RD JONES.

RD is a reserved word and its use causes unpredictable results.

3. All Procedure Division paragraph-names must start in Column 8.
4. No COPY verbs may appear in procedures on the library.
5. Names in the Data Division must not be the same as paragraph-names in the Procedure Division. For example, if the library contained:

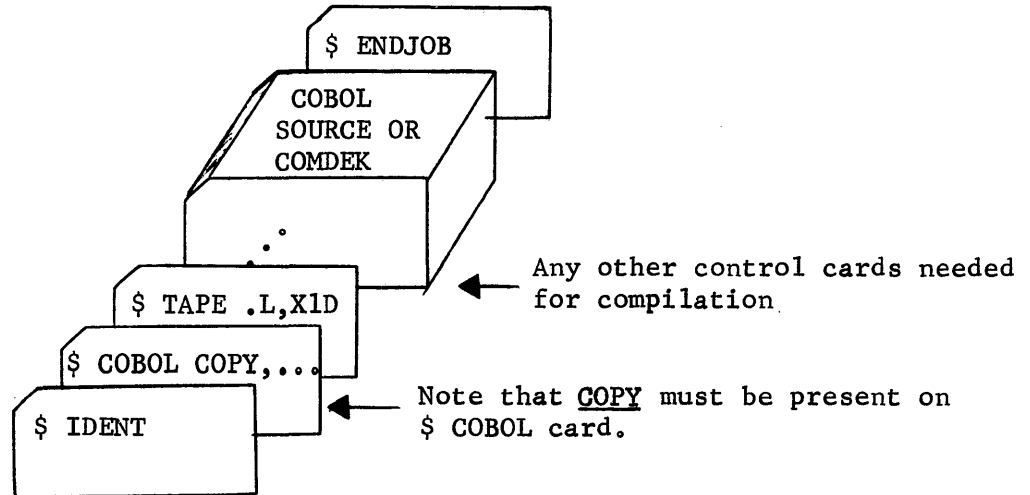
```
FD MASTER-UPDATE...  
.  
.  
.  
PROCEDURE DIVISION.  
MASTER-UPDATE. ...  
.  
.  
.
```

a COPY paragraph-name for MASTER-UPDATE might result in the insertion of the FD entry in the Procedure Division.

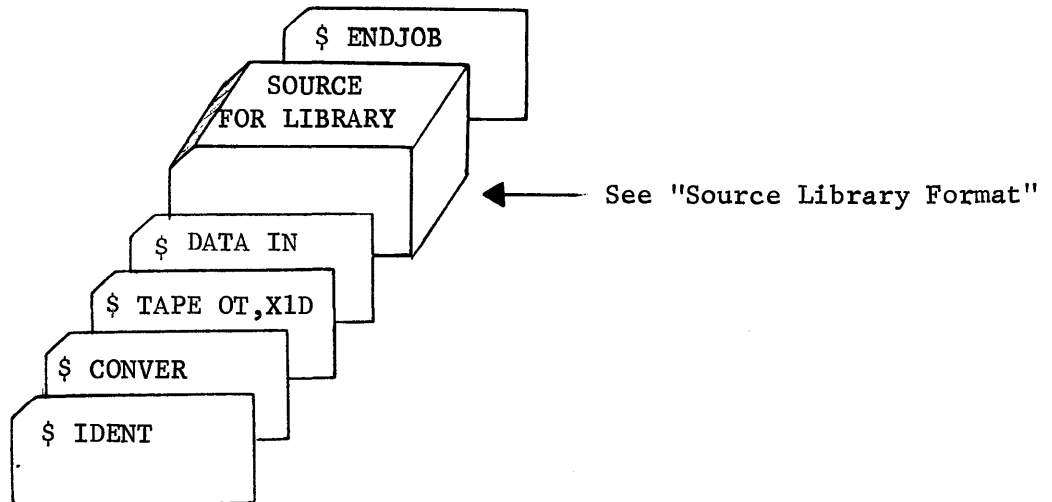
Deck Setups

Examples of deck setups for the creation and use of a library file are shown below.

The 2 examples below may be combined into a 2-Activity Job in the order 1. followed by 2.



2. Compilation using a library file



1. Creating a library file (file code = .L)

INDEX

This key-word index is formed by permuting titles, paragraph names, descriptive phases, and figure titles, putting each key word in the index position in the center of the page. The rest of the phrase appears on either side of the key word.

ABBREVIATIONS	VII	12, 13, 14
ACCEPT	VII	21
ACTUAL DECIMAL POINT	VI	34
ADD	VII	22
ADVANCING OPTION	VII	70
AE - ALPHANUMERIC EDITED	VI	33
ALGEBRAIC SIGNS	VI	19
ITEM ALIGNMENT AND SPACING	VI	19
ALPHABETIC	VI	22
ALPHANUMERIC	VI	22
ALPHANUMERIC CHARACTERS	VI	33
AE - ALPHANUMERIC EDITED	VI	33
ALTER	VII	24
RESERVE ALTERNATE AREA	E	1
ALTERNATE ENTRY SYMBOLS FOR GEFRM MODULES	D	2
APPLY BLOCK SERIAL NUMBER	VIII	7
APPLY PROCESS AREA	E	1
APPLY PROCESS AREA	VIII	7
APPLY SYSTEM STANDARD OR VLR FORMAT ON STATE	VIII	7.1
PROCESS AREA	E	1
SAME AREA	B	33
APPLY PROCESS AREA	E	1
SAME AREA	E	1
PROCESS AREA	B	1, 33
RESERVE ALTERNATE AREA	E	1
PROCESS AREA	B	1, 33
APPLY PROCESS AREA	VIII	7
FILE STORAGE AREAS	B	32
CONSTANT SECTION STORAGE AREAS	VIII	4.1
ASCENDING CLAUSE	VII	62
ASSIGNED FILE-CODES	VIII	6
ASSUMED DECIMAL SCALING POSITION	VI	32
ASSUMED DECIMAL POINT	VI	32
ASTERISK REPLACEMENT CHARACTER	VI	35
BASIC OPERATORS	VII	18
THE BCD AND LOW-DENSITY OPTIONS	B	15
BEGINNING-FILE-LABEL	B	13
STANDARD FLOATING-POINT BINARY FORMAT	A	3
MINIMUM BINARY SCALE	A	11
BITS OPTION	VI	38
BLANK WHEN ZERO OPTION	VI	26
APPLY BLOCK SERIAL NUMBER	VIII	7
BLOCK SERIAL NUMBER	B	9
BLOCK SIZE	VI	6
BLOCK SIZE	B	6

	BRACES	IV	1
	BRACKETS	IV	1
FINAL TYPE CONTROL	BREAK	VI	78
	PAGE BREAK DEFINITION	VI	53
CONTROL FOOTING OR	CE	VI	78
CONTROL HEADING OR	CH	VI	77
	SPECIAL INSERTION CHARACTER	VI	34
	DOLLAR SIGN CHARACTER	VI	35
ASTERISK REPLACEMENT	CHARACTER	VI	35
	COMPLETE COBOL CHARACTER SET	V	1
EDITING SIGN CONTROL	CHARACTERS	VI	34
	FIXED INSERTION CHARACTERS	VI	34
	ALPHANUMERIC CHARACTERS	VI	33
	CHECK PROTECT	VI	26
	CLASS	VI	22,63
	CLASS TEST	VII	9
	PICTURE CLAUSE	VI	27
PLACES OPTION OF THE POINT	CLAUSE	VI	38
NAMES INTRODUCED BY A RENAMES	CLAUSE	VI	18
	LINE-NUMBER CLAUSE	VI	62
	VALUE CLAUSE	VI	87
RESULT OF SYNCHRONIZED	CLAUSE	B	26
	ASCENDING CLAUSE	VII	62
	SAME CLAUSE	VIII	8
	DESCENDING CLAUSE	VII	62
	CURRENCY SIGN CLAUSE	VIII	4.2
	EDITING CLAUSES	VI	26,67
	CLOSE	VII	25
PROCESSING STRANGER FILES VIA	COBOL	D	1
	QUALIFICATION IN COBOL	V	7
	COMPLETE COBOL CHARACTER SET	V	1
	COBOL FILE FORMATS	B	1
	CODE	VI	56
	COLLATE COMMERCIAL	VIII	4.2
COMMERCIAL	COLLATING SEQUENCE	VII	8
STANDARD	COLLATING SEQUENCE	VII	7
	COLUMN DEFINITION	VI	52
	COLUMN NUMBER	VI	64
DECIMAL-POINT IS	COMMA	VIII	4.2
	COLLATE COMMERCIAL	VIII	4.2
	COMMERCIAL COLLATING SEQUENCE	VII	8
RECURRENT STRANGER	COMMUNICATIONS	D	6
	DATA FILE COMMUNICATIONS	X	1
	PROCEDURAL COMMUNICATIONS	X	3
	WORKING-STORAGE COMMUNICATIONS	X	2
	COMPARISON OF NUMERIC ITEMS	VII	5
	COMPARISON OF NON-NUMERIC ITEMS	VII	6
	COMPILER DIRECTING DECLARATIVES	VII	18
	COMPILER DIRECTING SENTENCES	VII	2,4
	COMPILER DIRECTING STATEMENTS	VII	2
	COMPLETE COBOL CHARACTER SET	V	1
FILE DESCRIPTION	COMPLETE ENTRY	VI	4
	COMPLETE ENTRY SKELETON	VI	21
	COMPLETE LIST OF RESERVED WORDS	V	11
	FORMING COMPOUND CONDITIONS	VII	17
	COMPOUND CONDITIONS	VII	10
	COMPUTATIONAL ITEM FORMATS	A	1
	COMPUTATIONAL ITEMS	A	3
	COMPUTATIONAL-1	E	3
USAGE	COMPUTATIONAL-1	VI	30
	COMPUTATIONAL-1	A	1
	COMPUTATIONAL-1 ITEMS	A	8
	COMPUTATIONAL-2	E	3

	COMPUTATIONAL-2	A	1
	COMPUTATIONAL-2 ITEMS	A	8
	COMPUTATIONAL-3	A	1
	COMPUTATIONAL-3 AND COMPUTATIONAL-4 ITEMS	A	9
	COMPUTATIONAL-4	A	1
	COMPUTATIONAL-4 DOUBLE PRECISION	A	10
COMPUTATIONAL-3 AND	COMPUTATIONAL-4 ITEMS	A	9
RESULT OF	COMPUTATIONAL USAGES	B	28
	COMPUTE	VII	28
	COMPUTER INDEPENDENT DETAILED DATA DESCRIPTION	VI	19
	CONCEPT OF LEVELS	VI	17
OVERFLOW	CONDITION	VI	79
	CONDITIONAL SENTENCES	VII	2,4
	CONDITIONAL STATEMENTS	VII	1
	CONDITIONAL VARIABLE TEST	VII	9
	CONDITIONALS	VII	5
SPECIFIC ENTRY FOR A	CONDITION-NAME	VI	48
	CONDITION-NAME	V	2
	CONDITION-NAMES	VI	87
COMPOUND	CONDITIONS	VII	10
FORMING COMPOUND	CONDITIONS	VII	17
	CONDITIONS	VII	5
	CONFIGURATION SECTION	VIII	1
	CONFIGURATION SECTION SOURCE-COMPUTER	VIII	2
	CONNECTIVES	IV	1
	CONNECTIVES	V	6
SERIES	CONNECTIVES	V	10
INTERNAL FORMAT	CONSIDERATIONS	B	24
EXTERNAL FORMAT	CONSIDERATIONS	B	2
SIZE OF A FIGURATIVE	CONSTANT	V	5
	CONSTANT RECORDS	VI	89
	CONSTANT SECTION	VI	88
WORKING-STORAGE OR	CONSTANT SECTION STORAGE AREAS	VIII	4.1
NON-CONTIGUOUS	CONSTANT STORAGE	VI	88
TABLE OF	CONSTANTS	VI	89
VALUE OF	CONSTANTS	VI	89
FIGURATIVE	CONSTANTS	V	4
FINAL TYPE	CONTROL BREAK	VI	78
EDITING SIGN	CONTROL CHARACTERS	VI	6
MINOR, INTERMEDIATE, AND MAJOR	CONTROL SPECIFICATIONS	VI	79
	FILE CONTROL CARDS	B	18
	I-O CONTROL	VIII	7
	CONTROL BREAK DEFINITION	VI	52
	CONTROL FOOTING OR CE	VI	78
	CONTROL GROUP DEFINITION	VI	52
	CONTROL HEADING OR CH	VI	77
	CONTROL HEADING REPORT GROUPS	VI	79
	CONTROL HIERARCHY DEFINITION	VI	52
	CONTROL(S)	VI	57
FILE	CONVERSION TO GE-625/635 FORMATS	D	5
	COPY	VI	24
	COPY	VI	58
	COPY	VI	65
	COPY	VI	7
	COUNTER DEFINITION	VI	53
	CURRENCY SIGN CLAUSE	VIII	4.2
COMPUTER INDEPENDENT DETAILED	DATA DESCRIPTION	VI	19
	DATA DESCRIPTIONS	E	3
	DATA DIVISION	III	3
	DATA DIVISION GENERAL DESCRIPTION	VI	1
	DATA DIVISION ORGANIZATION	VI	1
	DATA FILE COMMUNICATIONS	X	1
STANDARD	DATA FORMAT	VI	19

SIZE OF THE DATA ITEM	VI	32
DATA MANIPULATION	E	2
DATA RECORDS	VI	8
DATA-NAME	VI	25,66
DATA-NAME	V	2
DATA-NAME DEFINITION	VI	52
DATA-NAMES INTRODUCED BY A RENAMES CLAUSE	VI	18
DATE-COMPILED	IX	3
DATE-WRITTEN	B	11
DETAIL OR DE	VI	78
ACTUAL DECIMAL POINT	VI	34
ASSUMED DECIMAL SCALING POSITION	VI	32
ASSUMED DECIMAL POINT	VI	32
DECIMAL-POINT IS COMMA	VIII	4.2
DECLARATIVES	VII	17
COMPILER DIRECTING DECLARATIVES	VII	18
DECLARATIVES	III	5
REPORT WRITER DEFINITIONS	VI	51,52,53
DEFINITION OF WORDS	V	2
ENTER DEFINITIONS	VII	35
DEPENDING OPTION	VI	30
DERIVATION OF EXTERNAL AND INTERNAL FORMAT	VI	19
DESCENDING CLAUSE	VII	62
RECORD DESCRIPTION	VI	21
INDEPENDENT DETAILED DATA DESCRIPTION	VI	19
RECORD DESCRIPTION	VI	16
DATA DIVISION GENERAL DESCRIPTION	VI	1
FILE DESCRIPTION COMPLETE ENTRY	VI	4
REPORT-GROUP DESCRIPTION ENTRY	VI	51
REPORT-NAME DESCRIPTION ENTRY	VI	50
RECORD DESCRIPTION ENTRY	V	8
FILE DESCRIPTION ENTRY	VI	2
DATA DESCRIPTIONS	E	3
WITHOUT THE PAGE-LIMITS LAST DETAIL OPTION	VI	80
DETAIL OR DE	VI	78
COMPUTER INDEPENDENT DETAILED DATA DESCRIPTION	VI	19
COMPILER DIRECTING DECLARATIVES	VII	18
DISPLAY	VII	29
DISTINCTION BETWEEN SUBSCRIPTS AND QUALIFIER	V	10
DIVIDE	VII	30
ENVIRONMENT DIVISION	VIII	1
PROCEDURE DIVISION	VII	1
IDENTIFICATION DIVISION	II	2
IDENTIFICATION DIVISION	IX	1
DATA DIVISION GENERAL DESCRIPTION	VI	1
DATA DIVISION ORGANIZATION	VI	1
FLOATING DOLLAR	VI	33
FLOAT DOLLAR SIGN	VI	26
DOLLAR SIGN CHARACTER	VI	35
DOUBLE PRECISION ITEMS	A	6,10
NE - NUMERIC EDITED	VI	33
AE - ALPHANUMERIC EDITED	VI	33
FE - FLOATING POINT NUMERIC EDITED	VI	36
REPORT EDITING	VI	20
EDITING CLAUSES	VI	67
EDITING CLAUSES	VI	26
EDITING SIGN CONTROL CHARACTERS	VI	34
ELEMENTARY ITEM DEFINITION	VI	52
ELEMENTARY ITEMS	VI	17
ENDING-TAPE-LABEL	B	12
ENTER	VII	31
ENTER DEFINITIONS	VII	35
ENTER LINKAGE MODE	VII	36

REPORT-GROUP ENTRIES	VI	61
RD ENTRY	VI	55
REPORT GROUP DESCRIPTION ENTRY	VI	51
REPORT NAME DESCRIPTION ENTRY	VI	50
FILE DESCRIPTION COMPLETE ENTRY	VI	4
RECORD DESCRIPTION ENTRY	V	8
FILE DESCRIPTION ENTRY	VI	2
SPECIFIC ENTRY FOR A CONDITION-NAME	VI	48
ENTRY FORMATS	VI	3,20,54
VERB AND ENTRY FORMATS	IV	1
COMPLETE ENTRY SKELETON	VI	21
ALTERNATE ENTRY SYMBOLS FOR GEFRC MODULES	D	2
ENTRY-COUNT	B	31
ENVIRONMENT DIVISION	VIII	1
ENVIRONMENT DIVISION	III	2
EXAMINE	VII	38
SENTENCE EXECUTION	VII	3
EXIT	VII	40
EXIT PARAGRAPH	X	3
DERIVATION OF EXTERNAL AND INTERNAL FORMAT	VI	19
EXTERNAL FORMAT CONSIDERATIONS	B	2
EXTERNAL VERSUS INTERNAL FORMATS	B	26
FE - FLOATING-POINT NUMERIC EDITED	VI	36
SIZE OF A FIGURATIVE CONSTANT	V	5
FIGURATIVE CONSTANTS	V	4
MULTIPLE FILE	VIII	8
DATA FILE COMMUNICATIONS	X	1
FILE CONTROL CARDS	B	18
FILE CONVERSION TO GE-625/635 FORMATS	D	5
FILE DESCRIPTION COMPLETE ENTRY	VI	4
FILE DESCRIPTION ENTRY	VI	2
COBOL FILE FORMATS	B	1
MULTIPLE FILE OPTION	B	15
FILE SECTION	VI	83
FILE SIZE	VI	6
FILE STORAGE AREAS	B	32
MULTIPLE FILE TAPES	B	15
ASSIGNED FILE-CODES	VIII	6
INPUT-OUTPUT SECTION FILE-CONTROL	VIII	5
USING FILE-NAME-2 OPTION	VII	63
FILE-SERIAL-NUMBER	B	10
NONLABELED MULTIPLE REEL FILES	C	1
PROCESSING STRANGER FILES VIA COBOL	D	1
KEY WORD FILLER	VI	25
FILLER ITEMS	VI	85
FINAL TYPE CONTROL BREAK	VI	78
FIXED INSERTION CHARACTERS	VI	34
FLOAT DOLLAR SIGN	VI	26
FLOATING DOLLAR	VI	33
FLOATING MINUS	VI	33
FLOATING PLUS	VI	33
STANDARD FLOATING POINT BINARY FORMAT	A	3
FE - FLOATING POINT NUMERIC EDITED	VI	36
WITH THE PAGE LIMITS FOOTING OPTION	VI	80
WITHOUT THE PAGE LIMITS FOOTING OPTION	VI	80
EXTERNAL AND INTERNAL FORMAT	VI	19
STANDARD DATA FORMAT	VI	19
SYSTEM STANDARD FORMAT	B	4,19
FLOATING-POINT BINARY FORMAT	A	3
LOGICAL RECORD FORMAT	B	3
EXTERNAL FORMAT CONSIDERATIONS	B	2
INTERNAL FORMAT CONSIDERATIONS	B	24
FORMAT DEFINITION	VI	52

SYSTEM STANDARD OR VLR FORMAT ON STATEMENT	VIII	7.1
RULES OF PROCEDURE FORMATION	VII	1
VERB FORMATS	VII	3
SENTENCE FORMATS	VII	3
ENTRY FORMATS	VI	3, 20, 54
VERB AND ENTRY FORMATS	IV	1
SPECIFIC FORMATS	VI	20
COMPUTATIONAL ITEM FORMATS	A	1
COBOL FILE FORMATS	B	1
TARGET FORMATS	D	2
EXTERNAL VERSUS INTERNAL FORMATS	B	26
FILE CONVERSION TO GE-625/635 FORMATS	D	5
FORMING COMPOUND CONDITIONS	VII	17
FORMULAS	VII	18
REPRESENTATION OF FRACTIONAL VALUES	A	2
FROM LIBRARY	VI	24
ALTERNATE ENTRY SYMBOLS FOR GEFRC MODULES	D	2
FILE CONVERSION TO GE-625/635 FORMATS	D	5
GENERATE	VII	41
GIVING OPTION	VII	64
GO	VII	43
RELATIVE NEXT GROUP	VI	53
NEXT GROUP	VI	62, 72
GROUP INDICATE	VI	68
GROUP-ITEM DEFINITION	VI	52
CONTROL-HEADING REPORT GROUPS	VI	79
SPECIFICATIONS AND HANDLING OF LABELS	VI	83
SPECIFICATION AND HANDLING OF LABELS	B	9
IDENTIFICATION	B	11
PURPOSE OF THE IDENTIFICATION DIVISION	II	2
IDENTIFICATION DIVISION	IX	1
IDENTIFICATION DIVISION	III	1
I-O CONTROL	VIII	7
IMPERATIVE SENTENCES	VII	2, 4
IMPERATIVE STATEMENTS	VII	1
COMPUTER INDEPENDENT DETAILED DATA DESCRIPTION	VI	19
GROUP INDICATE	VI	68
INITIAL VALUE OF ANY ITEM IN THE WORKING-STOR	VI	87
INITIATE	VII	44
INPUT PROCEDURE	VII	62
INPUT-OUTPUT SECTION	VIII	1
INPUT-OUTPUT SECTION FILE-CONTROL	VIII	5
SPECIAL INSERTION CHARACTER	VI	77
FIXED INSERTION CHARACTERS	VI	34
INSTALLATION	B	10
MINOR, INTERMEDIATE, AND MAJOR CONTROL SPECIFICATION	VI	79
DERIVATION OF EXTERNAL AND INTERNAL FORMAT	VI	19
INTERNAL FORMAT CONSIDERATIONS	B	24
EXTERNAL VERSUS INTERNAL FORMATS	B	26
DATA-NAMES INTRODUCED BY A RENAMES CLAUSE	VI	18
SIZE OF THE DATA ITEM	VI	32
SYNCHRONIZED ITEM	VI	20
ELEMENTARY ITEM ALIGNMENT AND SPACING	VI	19
COMPUTATIONAL ITEM DEFINITION	VI	52
COMPUTATIONAL ITEM FORMATS	A	1
INITIAL VALUE OF ANY ITEM IN THE WORKING-STORAGE	VI	87
OCCURS...DEPENDING ITEMS	VI	30
COMPARISON OF NUMERIC ITEMS	VII	5
FILLER ITEMS	VI	85
COMPARISON OF NON-NUMERIC ITEMS	VII	6
ELEMENTARY ITEMS	VI	17

COMPUTATIONAL-1 ITEMS	A	8
COMPUTATIONAL-3 AND -4 ITEMS	A	9
DOUBLE PRECISION ITEMS	A	6
COMPUTATIONAL ITEMS	A	3
COMPUTATIONAL-2 ITEMS	A	8
JUSTIFIED	VI	69
JUSTIFIED RIGHT	VI	28
KEY WORD FILLER	VI	25
KEY WORDS	V	6
LABEL RECORDS	VI	10
IF LABEL RECORDS ARE OMITTED	B	13
SPECIFICATIONS AND HANDLING OF LABELS	VI	83
SPECIFICATION AND HANDLING OF LABELS WITHOUT THE PAGE LIMITS	B	9
LAST DETAIL OPTION	VI	80
LEAVING OPTION	VI	26
SYNCHRONIZED LEFT	B	27
VLR VARIABLE LENGTH RECORD	VIII	7.1
LEVEL NUMBER	VI	29
LEVEL-NUMBER	VI	70
CONCEPT OF LEVELS FROM LIBRARY	VI	17
VI	VI	24
LINE DEFINITION	VI	52
RELATIVE LINE NUMBER	VI	53
LINE-COUNTER	VI	53
LINE NUMBER	VI	71
LINE NUMBER CLAUSE	VI	62
ENTER LINKAGE MODE	VII	36
COMPLETE LIST OF RESERVED WORDS	V	11
NUMERIC LITERAL	V	3
LITERALS	V	3
POINT LOCATION	VI	38
LOGICAL RECORD FORMAT	B	3
LOWER CASE WORDS	IV	1
THE BCD AND LOW DENSITY OPTIONS	B	15
MINOR, INTERMEDIATE, AND MAJOR CONTROL SPECIFICATIONS	VI	79
DATA MANIPULATION	E	2
PERFORM MECHANISM FOR OPTIONS 1 THROUGH 4	VII	53
THE PERIPHERAL MEDIUM	B	17
MINIMUM BINARY SCALE	A	11
MINOR, INTERMEDIATE, AND MAJOR CONTROL SPECIFIC	VI	79
FLOATING MINUS	VI	33
ENTER LINKAGE MODE	VII	36
RECORDING MODE	VI	12,30.1
ENTRY SYMBOLS FOR GEFRC MODULES	D	2
MOVE	VII	45
MULTILEVEL SUBSCRIPTS	V	9
MULTIPLE FILE	VIII	8
MULTIPLE FILE OPTION	B	15
MULTIPLE FILE TAPES	B	15
PROCESSING NONLABELED MULTIPLE REEL FILES	C	1
MULTIPLY	VII	48
NE - NUMERIC EDITED	VI	33
RELATIVE NEXT GROUP	VI	53
NEXT GROUP	VI	62,72
NON-CONTIGUOUS CONSTANT STORAGE	VI	88
NON-CONTIGUOUS WORKING-STORAGE	VI	86
COMPARISON OF NON-NUMERIC ITEMS	VII	6
PROCESSING NONLABELED MULTIPLE REEL FILES	C	1

	NOTE	VII	49
	NOUNS	V	2
	COLUMN NUMBER	VI	64
	RELATIVE LINE NUMBER	VI	53
	LEVEL NUMBER	VI	29
	BLOCK SERIAL NUMBER	B	9
	NUMERIC	VI	22
FE - FLOATING-POINT	NUMERIC EDITED	VI	36
	NE - NUMERIC EDITED	VI	33
	COMPARISON OF NUMERIC ITEMS	VII	5
	NUMERIC LITERAL	V	3
	OBJECT-COMPUTER	VIII	3
	OCCURS	VI	30
	RESULT OF OCCURS...DEPENDING	B	29
	OCCURS...DEPENDING ITEMS	VI	30
OVERFLOW-HEADING OR	OH	VI	77
	OPEN	VII	50
	BASIC OPERATORS	VII	18
PAGE LIMITS LAST DETAIL	OPTION	VI	80
PAGE LIMITS FOOTING	OPTION	VI	80
PAGE LIMITS FOOTING	OPTION	VI	80
	BITS OPTION	VI	38
	RENAMING OPTION	VIII	5
VARYING SUBSCRIPT-NAME	OPTION	VII	54
	GIVING OPTION	VII	64
	RESERVE OPTION	VIII	6
USING FILE-NAME-2	OPTION	VII	63
	ADVANCING OPTION	VII	70
	LEAVING OPTION	VI	26
BLANK WHEN ZERO	OPTION	VI	26
	DEPENDING OPTION	VI	30
	MULTIPLE FILE OPTION	B	15
	RERUN OPTION	VIII	7.1
	UNTIL OPTION OF PERFORM	E	3
PLACES	OPTION OF THE POINT CLAUSE	VI	38
	OPTIONAL WORDS	IV	1
	OPTIONAL WORDS	V	6
	PAGE LIMITS OPTIONS	VI	59
THE BCD AND LOW DENSITY	OPTIONS	B	15
PERFORM MECHANISM FOR	OPTIONS 1 THROUGH 4	VII	53
	OUTPUT PROCEDURE	VII	63
	OVERFLOW CONDITION	VI	79
	OVERFLOW FOOTING OR OV	VI	78
	OVERFLOW HEADING OR OH	VI	77
	PAGE BREAK DEFINITION	VI	53
	PAGE DEFINITION	VI	51
	PAGE COUNTER	VI	54
	PAGE FOOTING OR PF	VI	78
	PAGE HEADING OR PH	VI	77
	WITH THE PAGE LIMITS FOOTING OPTION	VI	80
	WITHOUT THE PAGE LIMITS FOOTING OPTION	VI	80
	WITHOUT THE PAGE LIMITS LAST DETAIL OPTION	VI	80
	PAGE LIMITS OPTIONS	VI	59
	EXIT PARAGRAPH	X	3
	PERFORM	VII	51
UNTIL OPTION OF	PERFORM	E	3
	PERFORM MECHANISM FOR OPTIONS 1 THROUGH 4	VII	53
	USE OF PERFORM STATEMENTS	VII	37
	PERIOD	IV	1
	THE PERIPHERAL MEDIUM	B	17
	STRANGER PERIPHERALS	D	6
PAGE-FOOTING OR	PF	VI	77

PAGE-HEADING OR PH	VI	78
PICTURE	VI	73
PICTURE	VI	31
PICTURE CLAUSE	VI	27
UTILIZATION OF PLUG-IN POINTS	D	4
PLUG-IN-POINTS	D	3
FLOATING PLUS	VI	33
POINT CLAUSE	VI	38
POINT LOCATION	VI	38
POINT LOCATION	VI	74
UTILIZATION OF PLUG-IN POINTS	D	4
ASSUMED DECIMAL SCALING POSITION	VI	32
COMPUTATIONAL-4 DOUBLE PRECISION	A	10
DOUBLE PRECISION ITEMS	A	6
PRINT GROUP DEFINITION	VI	52
PROCEDURAL COMMUNICATIONS	X	3
INPUT PROCEDURE	VII	62
OUTPUT PROCEDURE	VII	63
PROCEDURE DIVISION	VII	1
RULES OF PROCEDURE FORMATION	VII	1
PROCEDURE DIVISION	III	4
PROCEDURE-NAME	V	3
APPLY PROCESS AREA	VIII	7
PROCESS AREA	E	1
APPLY PROCESS AREA	E	1
PROCESS AREA	B	1
PROCESS AREA	B	33
PROCESSING NONLABELED MULTIPLE REEL FILES	C	1
PROCESSING STRANGER FILES VIA COBOL	D	1
PROGRAM-ID	X	3
PROGRAM-ID	IX	2
CHECK PROTECT	VI	26
SENTENCE PUNCTUATION	VII	3
PURPOSE OF THE IDENTIFICATION DIVISION	II	2
QUALIFICATION IN COBOL	V	7
BETWEEN SUBSCRIPTS AND QUALIFIERS	V	10
RANGE	VI	39
RD ENTRY	VI	55
READ	VII	58
RECORD DESCRIPTION	VI	16
RECORD DESCRIPTION	VI	21
RECORD DESCRIPTION ENTRY	V	8
RECORD SIZE	VI	11
RECORD SIZE	VI	11
RECORDING MODE	VI	12,30
VLR VARIABLE LENGTH RECORD	VIII	7.1
WORKING-STORAGE RECORDS	VI	87
CONSTANT RECORDS	VI	89
DATA RECORDS	VI	8
LABEL RECORDS	VI	10
LABEL RECORDS	VI	10
RECURRENT STRANGER COMMUNICATIONS	D	6
REDEFINES	VI	40
PROCESSING NONLABELED MULTIPLE REEL FILES	C	1
REEL-NUMBER	B	10
REEL-SERIAL-NUMBER	B	10
REFERENCE FORMAT	III	1
RELATION TESTS	VII	5
RELATIVE LINE NUMBER	VI	53
RELATIVE NEXT GROUP	VI	53
RELEASE	VII	60
RELEASE AND RETURN STATEMENTS	VI	2

	RENAMES	VI	41
DATA-NAMES INTRODUCED BY A	RENAMES CLAUSE	VI	18
	RENAMING OPTION	VIII	5
ASTERISK	REPLACEMENT CHARACTER	VI	35
	REPORT	VI	50
	REPORT DEFINITION	VI	51
	REPORT EDITING	VI	20
CONTROL-HEADING	REPORT GROUPS	VI	79
	REPORT SECTION	VI	50
	REPORT FOOTING OR RF	VI	78
	REPORT-GROUP DEFINITION	VI	52
	REPORT GROUP DESCRIPTION ENTRY	VI	51
	REPORT GROUP ENTRIES	VI	61
	REPORT HEADING OR RH	VI	77
	REPORT NAME DESCRIPTION ENTRY	VI	50
	REPORTS%S	VI	13
	REPRESENTATION OF FRACTIONAL VALUES	A	2
	REQUIRED WORDS	IV	1
	RERUN OPTION CAUSES	VIII	7.1
	RESERVE ALTERNATE AREA	E	1
	RESERVE OPTION	VIII	6
	RESERVED WORDS	V	6
COMPLETE LIST OF	RESERVED WORDS	V	11
	RESULT OF COMPUTATIONAL USAGES	B	28
	RESULT OF OCCURS...DEPENDING	B	29
	RESULT OF SYNCHRONIZED CLAUSE	B	26
	RETENTION-PERIOD	B	11
	RETURN	VII	61
RELEASE AND	RETURN STATEMENTS	VI	2
REPORT FOOTING OR	RF	VI	78
REPORT HEADING OR	RH	VI	77
JUSTIFIED	RIGHT	VI	28
	RULES OF PROCEDURE INFORMATION	VII	1
	SAME AREA	B	33
	SAME AREA	E	1
	SAME CLAUSE	VIII	8
	SAMPLE-RECORD	B	31
MINIMUM BINARY	SCALE	A	11
ASSUMED DECIMAL	SCALING POSITION	VI	32
WORKING-STORAGE	SECTION	VI	86
CONSTANT	SECTION	VI	88
FILE	SECTION	VI	83
REPORT	SECTION	VI	50
INPUT-OUTPUT	SECTION	VIII	1
CONFIGURATION	SECTION	VIII	1
INPUT-OUTPUT	SECTION FILE-CONTROL	VIII	5
CONFIGURATION	SECTION SOURCE-COMPUTER	VIII	2
WORKING-STORAGE OR CONSTANT	SECTION STORAGE AREAS	VIII	4.1
SOURCE-PROGRAM	SEGMENTATION	X	1
	SELECT SENTENCES	VIII	5
	SENTENCE EXECUTION	VII	3
	SENTENCE FORMATS	VII	3
	SENTENCE PUNCTUATION	VII	3
COMPILER-DIRECTING	SENTENCES	VII	2,4
	SENTENCES	VII	2
CONDITIONAL	SENTENCES	VII	2,4
IMPERATIVE	SENTENCES	VII	2,4
SELECT	SENTENCES	VIII	5
	SEQUENCED	VI	14
BLOCK	SERIAL NUMBER	B	9
	SERIES CONNECTIVES	V	10
	SIGN	VI	42
FLOAT DOLLAR	SIGN	VI	26

	DOLLAR SIGN CHARACTER	VI	35
	EDITING SIGN CONTROL CHARACTERS	VI	34
	ALGEBRAIC SIGNS	VI	19
	SIZE	VI	43,74
	BLOCK SIZE	B	6
	BLOCK SIZE	VI	6
	FILE SIZE	VI	6
	RECORD SIZE	VI	11
	RECORD SIZE	VI	11
	SIZE OF A FIGURATIVE CONSTANT	V	5
	SIZE OF THE DATA ITEM	VI	32
	COMPLETE ENTRY SKELETON	VI	21
	SORT	VII	62
	THE SORT VERB	VI	2
	SORT-FILE	VI	2
	CONFIGURATION SECTION SOURCE-COMPUTER	VIII	2
	SOURCE-PROGRAM SEGMENTATION	X	1
	SOURCE-SUM-VALUE	VI	75
	ITEM ALIGNMENT AND SPACING	VI	19
	SPECIAL INSERTION CHARACTER	VI	34
	SPECIAL-NAMES	VIII	4
	SPECIFIC ENTRY FOR A CONDITION-NAME	VI	48
	SPECIFIC FORMATS	VI	20
	SPECIFICATION AND HANDLING OF LABELS	B	9
	INTERMEDIATE, AND MAJOR CONTROL SPECIFICATIONS	VI	79
	SPECIFICATIONS AND HANDLING OF LABELS	VI	83
	STANDARD COLLATING SEQUENCE	VII	7
	STANDARD DATA FORMAT	VI	19
	STANDARD FLOATING-POINT BINARY FORMAT	A	3
	SYSTEM STANDARD FORMAT	B	4
	SYSTEM STANDARD FORMAT	B	19
	APPLY SYSTEM STANDARD OR VLR FORMAT ON STATEMENT	VIII	7.1
	STANDARD ZERO SUPPRESSION	VI	35
	COMPILER DIRECTING STATEMENTS	VII	2
	CONDITIONAL STATEMENTS	VII	1
	IMPERATIVE STATEMENTS	VII	1
	USE OF PERFORM STATEMENTS	VII	37
	RELEASE AND RETURN STATEMENTS	VI	2
	SWITCH STATUS TESTS	VII	9.1
	STOP	VII	65
	NON-CONTIGUOUS CONSTANT STORAGE	VI	88
	CONSTANT SECTION STORAGE AREAS	VIII	4.1
	FILE STORAGE AREAS	B	32
	PROCESSING STRANGER FILES VIA COBOL	D	1
	STRANGER PERIPHERALS	D	6
	SUBSCRIPTING	V	8
	VARYING SUBSCRIPT-NAME OPTION	VII	54
	MULTILEVEL SUBSCRIPTS	V	9
	DISTINCTION BETWEEN SUBSCRIPTS AND QUALIFIERS	V	10
	SUBTRACT	VII	66
	ZERO SUPPRESS	VI	26
	STANDARD ZERO SUPPRESSION	VI	35
	SWITCH STATUS TESTS	VII	9.1
	ALTERNATE ENTRY SYMBOLS FOR GEFRC MODULES	D	2
	SYNCHRONIZED	VI	44
	SYNCHRONIZED	VI	30
	RESULT OF SYNCHRONIZED CLAUSE	B	26
	SYNCHRONIZED ITEM	VI	20
	SYNCHRONIZED LEFT	B	27
	SYNCHRONIZED RIGHT	B	26
	SYSTEM STANDARD FORMAT	B	4
	SYSTEM STANDARD FORMAT	B	19
	APPLY SYSTEM STANDARD OR VLR FORMAT ON STATEMENT	VIII	7.1

	TABLE OF CONSTANTS	VI	89
	TALLY	V	6
	UTILITY TAPES	VIII	6
	MULTIPLE FILE TAPES	B	15
	TARGET FORMATS	D	2
	TERMINATE	VII	67
	CLASS TEST	VII	9
	CONDITIONAL VARIABLE TEST	VII	9
	SWITCH STATUS TESTS	VII	9.1
	RELATION TESTS	VII	5
	TYPE	VI	77
	UNTIL OPTION OF PERFORM	E	3
	USAGE	VI	82
	USAGE	VI	45
	USAGE COMPUTATIONAL-1	VI	30
	USE	VII	68
	USE OF PERFORM STATEMENTS	VII	37
	USING FILE-NAME-2 OPTION	VII	63
	UTILITY TAPES	VIII	6
	UTILIZATION OF PLUG-IN POINTS	D	4
	VALUE	VI	76
	VALUE	VI	15
	VALUE	VI	47
	VALUE CLAUSE	VI	87
	INITIAL VALUE OF ANY ITEM IN THE WORKING-STORAGE	VI	87
	VALUE OF CONSTANTS	VI	89
	PRESENTATION OF FRACTIONAL VALUES	A	2
	CONDITIONAL VARIABLE TEST	VII	9
	VARYING SUBSCRIPT-NAME OPTION	VII	54
	THE SORT VERB	VI	2
	VERB AND ENTRY FORMATS	IV	1
	VERB FORMATS	VII	3
	VERBS	VII	20
	VERBS	V	6
	PROCESSING STRANGER FILES VIA COBOL	D	1
	APPLY SYSTEM STANDARD OR VLR FORMAT ON STATEMENT	VIII	7.1
	VLR VARIABLE LENGTH RECORD	VIII	7.1
	WITHOUT THE PAGE LIMITS FOOTING OPTION	VI	80
	WITHOUT THE PAGE LIMITS LAST DETAIL OPTION	VI	80
	KEY WORD FILLER	VI	25
	REQUIRED WORDS	IV	1
	OPTIONAL WORDS	IV	1
	LOWER-CASE WORDS	IV	1
	DEFINITION OF WORDS	V	2
	RESERVED WORDS	V	6
	OPTIONAL WORDS	V	6
	KEY WORDS	V	6
	COMPLETE LIST OF RESERVED WORDS	V	11
	VALUE OF ANY ITEM IN THE WORKING-STORAGE	VI	87
	NON-CONTIGUOUS WORKING-STORAGE	VI	86
	WORKING-STORAGE COMMUNICATIONS	X	2
	WORKING-STORAGE OR CONSTANT SECTION STORAGE	VIII	4.1
	WORKING-STORAGE RECORDS	VI	87
	WORKING-STORAGE SECTION	VI	86
	WRITE	VII	70
	BLANK WHEN ZERO OPTION	VI	26
	ZERO SUPPRESS	VI	26
	STANDARD ZERO SUPPRESSION	VI	35

STAPLE

STAPLE

FOLD

FIRST CLASS
PERMIT, No. 4332
PHOENIX, ARIZONA

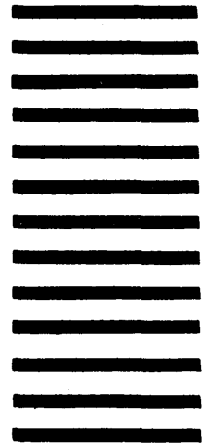
BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

GENERAL ELECTRIC COMPANY
INFORMATION SYSTEMS EQUIPMENT DIVISION
13430 NORTH BLACK CANYON HIGHWAY
PHOENIX, ARIZONA 85029

ATTENTION: DOCUMENTATION LARGE SYSTEMS DEPARTMENT C-77



FOLD

INFORMATION SYSTEMS

GENERAL  ELECTRIC