UNIVAC®

FUNDAMENTALS OF COBOL

LANGUAGE

PROGRAMMERS REFERENCE

.4

This manual is published by the Univac Division of Sperry Rand Corporation in loose leaf format. This format provides a rapid and complete means of keeping recipients apprised of UNIVAC® Systems developments. The information presented herein may not reflect the current status of the programming effort. For the current status of the programming, contact your local Univac Representative.

The Univac Division will issue updating packages, utilizing primarily a page-for-page or unit replacement technique. Such issuance will provide notification of software changes and refinements. The Univac Division reserves the right to make such additions, corrections, and/or deletions as, in the judgment of the Univac Division, are required by the development of its Systems.

PRINTED IN U.S.A.

CONTENTS

FIGURES

TABLES

1. INTRODUCTION

1.1. WHAT IS COBOL?

In attempting to establish what COBOL is, it is important to first establish how and why it came about. If any single word can be used to express its reason for being, that word is "communication". The need for improved communication has long existed between all elements concerned with the proper functioning of business, educational, and governmental organizations. Frequently, these various organizations did not speak the same language. Growth in complexity of these institutions has necessitated the introduction of a new tool – the computer – to more adequately fulfill the needs of each. The introduction of this device, though a valuable asset, added one more wall to the communications barrier. Now, not only was management having difficulty understanding the technician, and the analyst understanding the accountant, statistician or what have you, but all were having difficulty understanding the computer; or more to the point, in applying computer techniques to their own individual specialties and producing results that were meaningful to more than a select few.

Digital computers, as the name indicates, accept digital information and produce digital results – hardly meaningful to most without annotation. COBOL, however, is a coordinated effort to define a series of rules and procedures whereby problems can be submitted and solutions can be framed in a language – stylized English – acceptable to all those involved in a particular data processing application. Indeed, this COmmon Business Oriented Language brings one close to having a computer language tailor-made to one's own needs, since it permits identification of many program elements to be made in English, and in a form readily understandable by the casual observer.

The "word" in COBOL is the basic language element. As in the English language, COBOL contains many types of words with which meaningful thoughts are formed. "Meaningful" is the key descriptor in this or any language. In human discourse, a word is meaningful because, at some point in time, this word has been defined and associated with a particular concept or object. In COBOL, a word is meaningful either because the programmer has given it meaning by relating it to a specific quantity or quantities, or because it is an inherent or built-in part of the language. The COBOL language program employs many types of words. It uses *nouns* which are established by the programmer to name the various data elements upon which the program operates. It uses *verbs* which are supplied by the COBOL system to direct the manner in which the data is treated. It also uses certain selected words to improve the readability of the language or to complete the meaning of a sentence. For example, the following sentence illustrates these various words:

```
SUBTRACT DEDUCTIONS  FROM GROSS-PAY  GIVING NET-PAY.
```

Here, DEDUCTIONS, GROSS-PAY, and NET-PAY are nouns assigned by the programmer to name specific quantities. SUBTRACT is a verb supplied by the COBOL language to direct that particular arithmetic operation. FROM and GIVING are words included in the COBOL language to complete the meaning of the sentence. Often, such words as IS, ARE, and ON are included in statements to improve readability and may be omitted. This manual explains the use of the various words and the form they may take.

In summary then, the COBOL language is one part of a system which provides a method of stating computer problems and solutions in English. It comprises a basic set of English words and symbols used to define and create a program. It further permits the definition and naming of data according to the individual dictates of the user rather than the peculiarities of the computer. It forms a functional language that is largely independent of computer make or model.

1.2. HISTORY AND BACKGROUND

On May 28 and 29, 1959, a meeting was called in the Pentagon for the purpose of considering both the desirability and feasibility of establishing a common language for the programming of electronic computers in business data processing. Representatives from users, government installations, computer manufacturers, and other interested parties were present. There was almost unanimous agreement that the project was both desirable and feasible at that time. The concept of three committees or task forces was agreed upon. They were called the *Short Range, Intermediate Range*, and *Long Range* committees, with appropriate time scales. The Short Range group was composed of six manufacturers and three government representatives. This committee held its first meeting on June 23, 1959. At that time it was decided that the tasks of the committee fell into four general areas. Working groups were established as follows:

- Data Description
- Procedural Statements
- Application Survey
- Usage and Experience

The first two groups held frequent meetings and prepared proposals for consideration by the full committee which met August 18–21 and August 24–25 for the purpose of preparing a report to the Executive Committee. Materials developed as the result of the work of the other two groups were used later in the course of the development of COBOL. The report to the Executive Committee, submitted in September 1959, stated that the Short Range Committee felt it had prepared a framework upon which an effective common business language could be built. It was recognized that the report contained rough spots and needed additions. It further requested that the Short Range Committee be authorized to complete and publish the system by December 1959. It also requested that the Short Range Committee continue beyond December in order to monitor the implementation. Both these requests were approved.

The Committee held meetings between September 18 and November 7, 1959, and proceeded steadily in its task of resolving problems and completing the language. The name "COBOL", a COmmon Business Oriented Language, was adopted.

The COBOL System was reviewed and approved by the Short Range Committee during the week of November 16–20, 1959. Final editing and initial distribution of the report to the Executive Committee was accomplished December 17, 1959.

After acceptance by the Executive Committee of CODASYL (COnference on DAta SYstems Languages), the report was published April 1960 by the Government Printing Office as

> "COBOL – A report to the Conference on Data Systems Languages, Including Initial Specifications for a Common Business Oriented Language (COBOL) for Programming Electronic Digital Computers."

A Maintenance Committee was created by the Executive Committee of the CODASYL group to be responsible for initiating and reviewing recommended changes to keep COBOL up-to-date. This Maintenance Committee comprised user and manufacturer groups. The Maintenance Committee considered and agreed on a number of revisions to COBOL.

In order to devote concentrated attention to publishing a revised and updated "COBOL-1960", the Executive Committee created a Special Task Group. The product of this task group was the COBOL-1961 manual published in mid-1961. In mid-1963, the Maintenance Committee released COBOL-1961 Extended.

The organizations participating in the Maintenance Committee or the Special Task Group were:

Air Materiel Command, United States Air Force
Allstate Insurance Company
Bendix Corporation, Computer Division
The Boeing Company
Burroughs Corporation
Chase Manhattan Bank
Control Data Corporation
David Taylor Model Basin, Bureau of Ships, U. S. Navy
DuPont Company
General Electric Company
General Motors Corporation
International Business Machines Corporation
Lockheed Aircraft Corporation
Minneapolis-Honeywell Regulator Company
National Cash Register Company
Owens-Illinois Incorporated
Philco Corporation
Radio Corporation of America
Royal McBee Corporation
Space Technology Laboratories, Incorporated
Southern Railway Company
Standard Oil Company (N.J.)
Sylvania Electric Products, Inc.
Univac Division of Sperry Rand Incorporated
United States Steel Corporation
Westinghouse Electric Corporation

In January 1964, the COBOL Maintenance Committee was reorganized to provide a true industry group and to broaden its scope of activities. As the result of this reorganization, three subcommittees were established: the Language Subcommittee, the Evaluation Subcommittee, and the Publication Subcommittee. All three subcommittees report to the COBOL Committee.

The Language Subcommittee took over the functions of the Maintenance Committee, namely, the maintenance and further development of COBOL.

The Publication Subcommittee is charged with the production of offical COBOL publications and with liaison with the United States of America Standards Institute (USASI) as to the content of the COBOL Information Bulletin. The COBOL Information Bulletin is a collection of material relating to COBOL which is distributed to the COBOL community.

The Evaluation Subcommittee's task is the analysis and evaluation of compiler implementations and user surveys. This subcommittee provides information to the COBOL Committee regarding the use of COBOL.

In November 1965, the COBOL Committee released COBOL-Edition 1965.

In 1961, a portion of the Intermediate Range Committee was combined with the Long Range Committee to form the Development Committee. This committee was comprised of a Systems Group and a Language Structure Group. Although there were occasional joint meetings of these two groups, it was concluded that it would be more fruitful to operate these groups as two separate committees. Accordingly, the CODASYL Executive Committee in April 1965, approved the reorganization of these two groups and designated them as follows:

(a) The CODASYL Systems Committee (CSC) with the objective of developing a data systems language which uses general, easily understood language as the medium expression; is independent of machines; and can be used as a common language for and between all digital computer systems as well as a medium for human communication.

(b) The CODASYL Language Structures Committee (CLSC) with the objective of developing a unifying structure for the specification of data processing systems which places emphasis on the nature of the data processing results to be accomplished rather than on the sequence of steps required; results in a language that is convenient to use by people concerned with a variety of data processing problems rather than with computer programming; and is readily implementable on data processing machines. In July 1965, CLSC was dissolved at its own request.

The American Standards Association (ASA) Sectional Committee X3 for Computers and Information Processing was established in 1960 under the sponsorship of the Business Equipment Manufacturers' Association (BEMA). ASA X3 in turn, established the X3.4 Sectional Subcommittee to work in the area of common programming language standards. Subsequently, Working Group X3.4.4 for Processor Specifications and COBOL Standards was formed with the primary goal of the group being the production of an American COBOL standard.

On December 17, 1961, invitations to an organizational meeting of the X3.4.4 Working Group were sent to manufacturer and user groups who might be interested in participating in the development of an American COBOL standard. The first meeting of the X3.4.4 Working Group was held in New York City on January 15/16, 1963. In August 1966, the ASA became the United States of America Standards Institute (USASI).

Since that first meeting back in January 1963, many meetings have been held and on August 30, 1966, the X3.4.4 Working Group approved the content and format for a proposed USASI standard COBOL. The resultant proposed standard is based upon the elements contained in CODASYL COBOL-Edition 1965 and as such, represents a proper subset of the total COBOL language as defined in that publication.

1.3. BENEFITS DERIVED FROM USING COBOL

The following are some of the many advantages in using COBOL:

- Demonstrated Compatibility

  The capability of using a program run on one computer and, with minor modification, running it on another of a different make or model, is a proven fact. This reduces the reprogramming time considerably from the time required to recode applications written in an equipment-oriented language.

- Increased Communication

  Since COBOL uses English-like statements, the program is intelligible to all who understand the application. This makes available to the programming effort, the knowledge and efforts of many experts formerly excluded.

- Faster Programming

  Since the programmer is freed from much of the clerical detail found in equipment-oriented languages, the time required to program new or altered applications is reduced. Emphasis is shifted from the programming effort involved in detailed coding to problem definition and analysis.

- Increased Program Accuracy

  COBOL uses built-in standard conventions based on extensive user and systems experience to develop efficient coding techniques. Because these techniques are automatically introduced into the program, greater coding efficiency is achieved. The debugged coding segments at the user's command minimize the chance of program error and simplify design.

- Reduced Training Time

  New programming personnel can be trained to write productive programs in substantially less time than it takes to train them in equipment-oriented languages. With COBOL, it is not necessary to have a large staff of trained programmers. The more complicated, time-consuming techniques of computer coding need be taught only to those few people selected to become highly-skilled, career programmers.

- Reduced Programming Costs

  The ability to program a problem faster, and for the most part more accurately, reduces the cost of programming. Reprogramming costs are also reduced, since programs produced for one system may be modified to run on other systems without being entirely recoded.

These are just some of the benefits to be derived from the use of COBOL. As the user becomes more familiar with COBOL, many other advantages of the system become increasingly evident.

1.4. COBOL AND THE COMPUTER

Digital computers will accept and act upon only a fixed set of instructions expressed in a specific manner. However, though the number and sophistication of instructions may vary from computer to computer, certain general concepts remain the same. Each computer will have instructions to implement at least the following functions:

Data Movement
Arithmetic Operations
Decision Making
Input/Output Operations

Each of these functions may comprise many instructions and the format for their specification varies with the myriad makes and models of computers presently available on the market. Unlike the various equipment-oriented languages (oriented to a single computer or to a single family of computers) which cater to the peculiarities of the individual computer instruction repertoire, COBOL operates at a functional level. That is, it provides in English-language form, facilities for treating the above-stated functions in terms of the problem to be solved rather than in terms of the computer to be programmed.

Certainly, the COBOL language is not free-form English. It is stylized and, to that extent, restricted. Unlike the human brain, the computer cannot accept nuances of expression; it cannot discern vocal inflection; it cannot recognize physical gestures meant for emphasis. Therefore, with built-in restrictions governed by what the computer can and cannot accept, the COBOL language operates within certain well-defined boundaries established by very definite rules of procedure. However, any language, whether it is designed for computers or human beings, contains many rules for construction and use. These rules enhance the usefulness of the language as a vehicle for discourse and communication. The programmer must familiarize himself with the basic rules and procedures inherent in the COBOL language to gain proficiency in problem definition, analysis, and solution.

1.5. THE COBOL SYSTEM

The COBOL system actually comprises two main elements: the program and the program processor (compiler).

When we refer to the program, we are actually referring to two programs as follows:

■ The Source Program

That set of written entries and statements prepared by the programmer and designed to perform the following functions:

– Describe the data.

– Instruct the computer by a set of procedural steps on how to treat the data to effect a logical solution to the problem.

– Identify the program.

– Describe the equipment being used.

- The Object Program

  That set of coded instructions and data acceptable to the computer which carries through the logic expressed in the source program, together with the storage assignments for the data to be processed.

To translate a problem-oriented COBOL source program to a machine-oriented object program written in a code form acceptable to the specific computer, a COBOL program *processor* (compiler) is provided. The compiler is a program which analyzes the words and characters of a COBOL-language program and produces a new program in a code acceptable to the particular computer. From a single COBOL statement, the compiler may produce many computer instructions. The compiler is entered into the computer together with the source program. It then produces the object program to be run for problem solution.

When translating the source program to object program, the program processor may perform the following functions:

- Decoding

  The intended meaning of individual characters or groups of characters written in the source language are ascertained and converted into computer-acceptable instructions and data.

- Conversion

  Numerical information is converted from one number base to another (i.e., decimal to binary) and from fixed to floating point notation or vice versa, if required.

- Selection

  A required routine may be selected from a library of routines.

- Generation

  Required subroutines are generated from various parameters and skeletal coding specified in the source program.

- Allocation

  Actual storage locations are assigned in memory to the various program elements.

- Assembly

  Subroutines (supplied, selected, or generated) are integrated into the program.

- Recording

  Detailed information concerning the object program is recorded and may be printed.

1.6. COBOL MODULES AND LEVELS

In order to more effectively implement the uses for which COBOL was designed, the COBOL language has been revised to include nine functional processing modules. The new organization is oriented toward the functional processing concept while still retaining the old divisional structure (Identification, Environment, Data, and Procedure) within each module. These modules are as follows:

- Nucleus
- Table Handling
- Sequential Access
- Random Access
- Sort
- Report Writer
- Segmentation
- Library

Each functional processing module is divided into two or more levels. These levels provide a hierarchy within each functional processing module and in all cases, a lower level constitutes a proper subset of the next higher level within the module. This hierarchal modular structure, illustrated in Figure 1–1, enables the user to tailor his COBOL compiler such that he obtains only as much computing power as he needs for his particular application.

1.7. SCOPE OF MANUAL

This manual follows the structure of the language as defined by USASI and describes the nucleus, sequential access, segmentation, and library modules as applied to tape, card, and printer files (normally sequentially written and accessed files). (Table Handling, Sorting, Mass Storage and Random Access, Report Writer, and Glossary are described in the other manuals of this series.)

All discussions in this manual are oriented toward the highest level (level 2) which provides the fullest implementation of the COBOL language.

<table>
<tr><th colspan="8">FUNCTIONAL PROCESSING MODULES</th></tr>
<tr><th>NUCLEUS</th><th>TABLE HANDLING</th><th>SEQUENTIAL ACCESS</th><th>RANDOM ACCESS</th><th>SORT</th><th>REPORT WRITER</th><th>SEGMENTATION</th><th>LIBRARY</th></tr>
<tr><td rowspan="2">LEVEL 2</td><td>LEVEL 3</td><td rowspan="2">LEVEL 2</td><td>LEVEL 2</td><td>LEVEL 2</td><td>LEVEL 2</td><td>LEVEL 2</td><td>LEVEL 2</td></tr>
<tr><td rowspan="2">LEVEL 2</td><td rowspan="2">LEVEL 1</td><td rowspan="2">LEVEL 1</td><td rowspan="2">LEVEL 1</td><td rowspan="2">LEVEL 1</td><td rowspan="2">LEVEL 1</td></tr>
<tr><td rowspan="2">LEVEL 1</td><td rowspan="2">LEVEL 1</td></tr>
<tr><td>LEVEL 1</td><td>NULL</td><td>NULL</td><td>NULL</td><td>NULL</td><td>NULL</td></tr>
</table>

NOTE: Minimum COBOL System which can be implemented is Nucleus (Level 1) plus Table Handling (Level 1) plus Sequential Access (Level 1)

Figure 1-1. COBOL Language Hierarchy

2. PROGRAM ORGANIZATION

2.1. GENERAL

A source program must contain four basic elements in order to solve a problem:

- A description of the data to be processed.
- A set of procedures or operations establishing how the data is to be treated.
- A description of the equipment to be used.
- Information that will label and identify a particular application.

This chapter discusses these components of the source program and indicates the relationship between them. It also illustrates the organization of a COBOL program by providing a narrative English discussion of a sample application.

2.2. DESCRIBING THE DATA

The programmer must describe each element of data upon which the object program is expected to operate. The following information must be included:

- The name by which the program is to identify a particular datum.
- The class of data (i.e., is it alphabetic, numeric, or alphanumeric?).
- The length of the datum in terms of characters.
- The location of special symbols such as the currency sign, comma, decimal point, etc.
- The relative position of this datum within a record with respect to the other data to be operated upon.

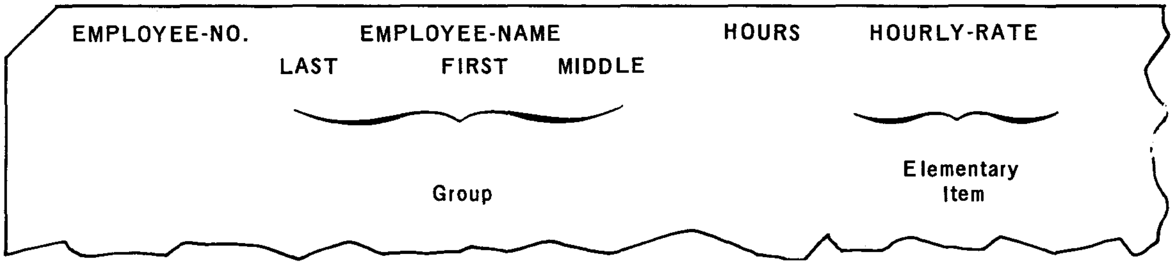
When the programmer provides this information, he is stating that, at object time, he will have an element of data to be processed and that the compiler should provide storage space based upon the requisites he has expressed. For example, suppose a programmer wishes to process data such as an employee's salary. First, he would assign a name to the datum, such as SALARY. Then he would state that the field is numeric (or alphanumeric if it contains characters other than digits), state the maximum size of the field, and show the position of both currency sign and decimal point. He would then show its relationship to the other data in the employee's record. The compiler, based on this information, would reserve a storage area for the data and, as the salary for each employee was fed in at object time, it would be placed in this location preparatory to being processed by the instructions in the program.

The COBOL language provides specific rules and procedures for supplying this information. Section 3 of this manual discusses them in detail.

The ensuing paragraphs describe the logical grouping and organization of data for a COBOL program.

2.2.1. Items and Groups

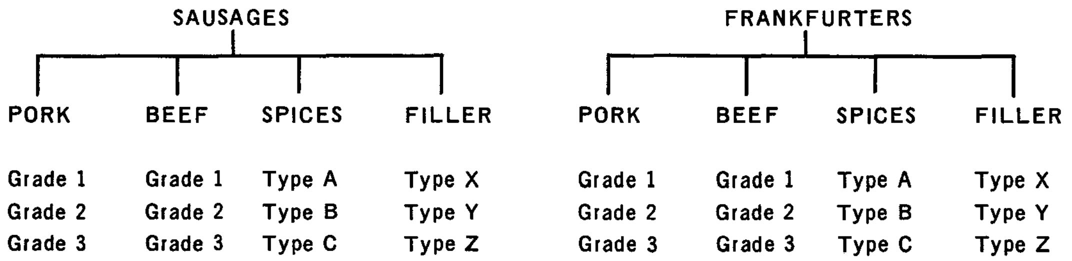
The smallest unit of data is the *elementary item*. This is a datum which is not further subdivided into smaller units when referenced in the program. The HOURLY-RATE field on an employee's weekly time card is an example of an elementary item; it is not broken into smaller units.



On the same weekly time card, there is a field called EMPLOYEE-NAME. This datum, however, is subdivided into smaller units called LAST, FIRST, MIDDLE. In this case, we are no longer dealing with a single unit of data, but with a *group* of data the individual components of which are elementary items relating to the entire group name EMPLOYEE-NAME.

When organizing data, it is also possible to have groups of groups. This is illustrated by the following example:

> Assume that the XYZ MEAT COMPANY maintains an inventory stock for the various ingredients necessary for the production of frankfurters and sausages. The data pertaining to these ingredients might be organized in the following manner:



GRADE and TYPE are elementary items and each designates a single quantity of the particular ingredient. However, PORK, BEEF, SPICES, and FILLER are groups comprising several related items. SAUSAGES and FRANKFURTERS are also groups. Instead of being groups of items, however, they are *groups* of *groups*.

To reiterate, elementary items are the smallest data units and are not further subdivided. Groups are larger data units comprising several elementary items or several groups.

2.2.2. Records

Related elementary items and groups are combined to form *records*. In the time card example, for instance, the card for each employee might constitute a single or *logical* record. Though the record format would remain the same, the values of the various elementary items would obviously change from employee to employee. The same holds true for the example of the XYZ MEAT COMPANY. In that example, the entire series of items and groups might be combined into one logical record called INVENTORY-ON-HAND.

Information for processing is read into the computer in complete records. It is not possible to read in only a part of a record. Similarly, only complete records may be written; that is, made available for output on some form of external medium such as magnetic tape or punched cards.

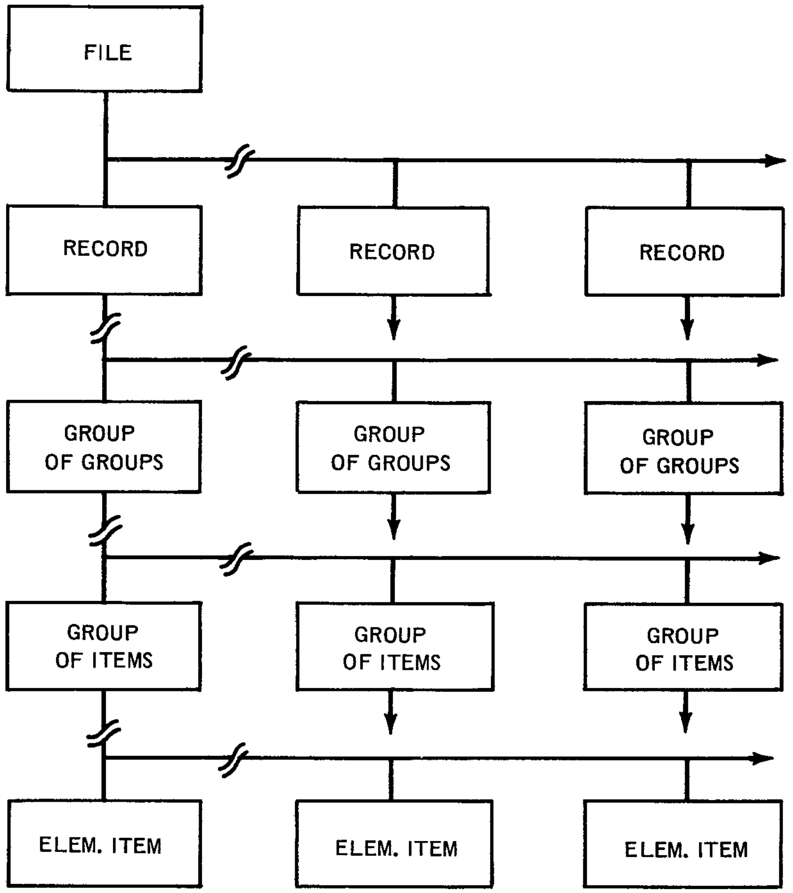
2.2.3. Files

A *file* is a set of records. Not all the records in a particular file need have the same format. When a record is read in for processing, it replaces the previous record of that file. If it is necessary to have more than one record of a particular file available to the program at a specific time, the first record must be moved to an intermediate work area before the second record may be read in. The exact method of doing this is discussed in subsequent sections.

The word ''file'' is used in COBOL because of the similarity to the file cabinet used in any office. For example, each drawer might be equated to a record in which is contained various types of related data. The file cabinet would also be labeled in some manner to indicate the contents. Similarly, a data processing file often has a label record to provide identification. Usually there is one label record appearing at the beginning of the file and another at the end. These label records serve as checking devices to ensure that the proper data is being read and that the file has been completed. There is no direct relationship between the length of a tape file and a reel of tape. A tape file may be contained on many reels of tape or a single tape may contain many files. Other files, such as printer files and card files, have their own unique characteristics.

2.2.4. Summary

The following diagram illustrates the hierarchal structure used in COBOL to describe data.



Each file consists of a set of records. There may be one type of record or several types of records. Each type of record is usually described as a set of elementary items; however, a record itself may be treated as an elementary item if it is not further subdivided.

It is often convenient to group two or more elementary items. These groups may then be further grouped into groups of groups, and so forth.

2.3. STATING THE PROCEDURES FOR PROBLEM SOLUTION

Once the data has been described and organized, the programmer may concentrate on the various operations or procedures necessary to solve his problem using the four major functions available to him:

- Data Movement
- Arithmetic
- Decision-Making
- Input/Output

In addition, the programmer has at his command Control functions which enable him to alter the sequence of program operation or cause repetitive cyclings of certain program steps.

When establishing the various procedures for problem execution, the programmer need not be aware of the internal details of the specific computer concerned in his application. He may concentrate fully on problem logic.

The various functions operate as follows:

- Input/output functions permit communication with the external media, such as magnetic tapes or punched card equipment, to obtain the specific records that are to be processed or to deliver final records to output.

- Decision-making capabilities permit comparisons of data to determine which of several possible operations are to be performed.

- Arithmetic functions permit the necessary calculations to be made on the data.

- Control functions permit selected operations to be performed or repeated a specified number of times.

- Data-movement instructions permit the movement or assembling of data into groupings prior to producing them on some output medium or performing calculations.

When setting up these procedures, a complete knowledge of the capabilities of the equipment is helpful from the standpoint of creating an efficient object program. The extent of the programmer's familiarity with the characteristics of his particular equipment is left to the discretion of the individual user. The specific rules and formats for procedure formation are covered in Section 4 of this manual.

2.4. IDENTIFYING THE PROGRAM

Having described the data and established the procedures for problem execution, the programmer will identify the particular application. He might wish to include such things as the date the program was written, the programmer's name, the name of the particular installation, and so forth. The mechanics by which this information is entered into the COBOL program are covered in detail in Section 6.

2.5. DESCRIBING THE EQUIPMENT

A description of the equipment is necessary to the proper operation of a COBOL program. Information such as the descriptions of the computers to be used for both compilation and object running of the program must be provided. This could include memory size and descriptions of the relationship between the logical records and their physical format on the input/output media. Problem-oriented names may be assigned to specific equipment to facilitate referencing in the program. This section, because it deals with specifications of the equipment being used, is largely computer-dependent. Details for providing this information are given in Section 5.

2.6. A SAMPLE APPLICATION

A typical data processing problem is presented here using common English sentences to provide the following information:

- A general definition of the problem or application.
- A general description of the data.
- A general description of the actions which will be performed on the data.
- A description of the manner in which the computing equipment is described.

Using this information as a guide, a parallel COBOL solution is developed throughout the manual as the individual elements comprising the COBOL language are discussed. The final solution is shown in full at the end of the manual. There are many ways of approaching any data processing problem, and the following is not intended to represent an optimum solution. *The purpose of this problem is to illustrate the use of as many basic COBOL facilities as is practical, while providing a comparison of the COBOL compiler language to the English language.*

2.6.1. Definition of Application

A sporting goods retailing chain updates its central warehouse inventory file on a daily basis to assure an adequate supply of goods for its many local dealers.

Each change of stock level on a stock item is recorded on a punched card, called a detail transaction. All detail transactions are then collected and sorted or arranged into order by stock item number and transaction code. This detail transaction file is run against the existing master inventory file to build an updated master inventory file.

As the updated master inventory file is created, a list is made of all stock items that have fallen below their minimum stock levels, and of new stock items which have been entered into the file on that particular update run. This list is called the stock replenishment or stock reorder list.

2.6.2. Description of Data

Two input files and two output files are used: the input master inventory file, the input detail transaction file, the output (updated) master inventory file, and the output stock reorder list file. In addition to the input and output files, certain other data items must be defined for intermediate arithmetic or logical results and for constant information, such as page headings. Detailed descriptions and examples of this information will be presented in the continuation of the sample problem discussion in Section 3.

2.6.3. Procedural Steps

The data having been described, the actions to be taken at object time are specified. The program begins by opening the files (i.e., initializing each file to prepare it for releasing or accepting data). Then data is read in stock number sequence, and the various manipulative, logical, and arithmetic operations performed. This done, the files are closed and the run terminated.

A detailed description of the procedural steps, including a flow chart, is given at the continuation of the sample problem discussion in Section 4.

2.6.4. Physical Environment Description

This section of the program describes those aspects of the total problem which depend upon the characteristics of the computing equipment.

The following information must be considered:

(a) The computer on which the compilation will be performed.

(b) The computer on which the compiled object program will be executed.

(c) Special mnemonic names which will be defined as equivalent to standard hardware names.

(d) Hardware assignments for the several files.

(e) Data transfer, including buffering, between the computer memory and the hardware media.

This information is discussed in detail in the sample problem discussion in Section 5.

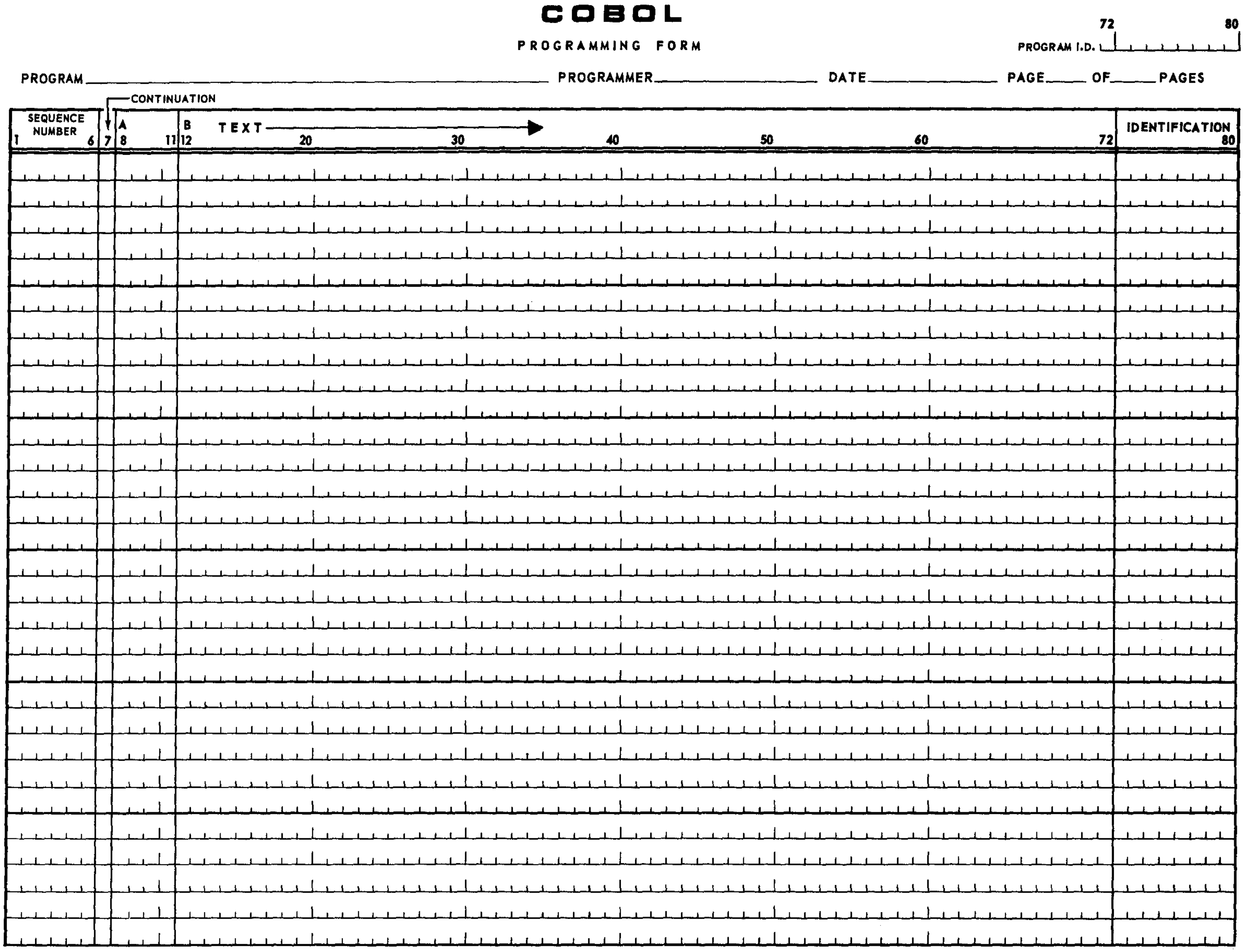
2.6.5. Program Identification

This section of the program identifies or labels the program. It may also contain any other information desired as to authorship, date of writing or compiling, security, and any other comments regarding the functional or peripheral aspects of the program.

Detailed information on program identification is given in the sample problem discussion in Section 6.

2.7. THE CODING FORM

Figure 2–1 shows the layout of a typical COBOL programming form. On this form the programmer enters all information needed by the COBOL compiler, observing certain rules of format and content as defined in this manual. Each line of written information represents the information to be entered into one 80-column punched card. The accompanying table explains the several divisions of the form.



COBOL

PROGRAMMING FORM

PROGRAM I.D. 72 80

PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ____ OF ____ PAGES

CONTINUATION

SEQUENCE NUMBER 1 6	7	A 8 11	B TEXT → 12 20 30 40 50 60 72	IDENTIFICATION 80

Figure 2–1. The COBOL Programming Form

COLUMNS	DESIGNATION	CONTENTS
1–6	SEQUENCE NUMBER	A numeric entry; used only by the programmer (not the COBOL processor) to establish a sequence among the various lines of coding (optional).
7	CONTINUATION	A hyphen (-); used when an entry extending past one line has a break occurring in the middle of a word. The hyphen is written in column 7 of the next contiguous line on which the word is completed. A word may be interrupted in any column, the rest of the line space filled, and completed on the next line. Parentheses are considered punctuation and do not require a hyphen in column 7 when they are split from the word they surround.
8–72	TEXT	All COBOL-formatted information, in the form of names, statements, information, instructions, etc., that is to be compiled into the object program. Note that two left-margin limits designated "A" and "B" are shown. These are needed for program alignment. Major definitive names are begun at margin A (column 8). Margin B (column 12) is used for subordinate items and for continuations of entries from the last preceding line.
73–80	IDENTIFICATION	Card deck information (optional).

A more complete explanation of the use of the form is presented in Section 7.

2.8. SYMBOLS, RULES, AND NOTATIONS USED IN THIS MANUAL

The various language elements that comprise a COBOL program must be written in formats that adhere to fixed and precise rules of presentation. Before discussing these individual formats, it is necessary to understand the various symbols, rules, and notations used in describing them. Each format statement will indicate the following information:

- The order of presentation.
- Those words that are requisite to the proper functioning of the statement.
- Those words that are optional and included at the discretion of the user.
- That information that must be supplied by the user.
- Those elements in the statement that involve a choice by the user.
- Those functions of the particular statement that are optional.

In free form, the MULTIPLY statement might appear in the following manner:

> Multiply a data name or a literal by another data name with the result rounded; on size error execute an imperative statement.

This, of course, tells us something about the order of presentation, but very little else about the rest of the format. Let us then establish the first rule of format presentation.

(a) All words inherent or built into the COBOL language are specified as uppercase.

> MULTIPLY a data name or a literal BY another data name with the result ROUNDED; ON SIZE ERROR execute an imperative statement.

(b) All uppercase words which are underlined are *required* or *key words*. Those uppercase words not underlined are *optional* and may be included at the user's discretion to improve readability.

$$\underline{\text{MULTIPLY}}\text{ a data name or a literal }\underline{\text{BY}}\text{ another data name with the result }\underline{\text{ROUNDED}};\text{ ON }\underline{\text{SIZE}}\ \underline{\text{ERROR}}\text{ execute an imperative statement.}$$

All uppercase words, whether underlined or not, are a part of the COBOL language and must be spelled exactly as indicated.

(c) All lower case words in italics represent generic terms which must be supplied by the user. In the sample statement, there are four such elements to be supplied by the user: two data names (which shall be designated *data-name-1* and *data-name-2* in order of their appearance) or a *literal,* and an *imperative-statement.*

(d) Elements of a statement involving a choice are surrounded by braces { }.

$$\underline{\text{MULTIPLY}}\begin{Bmatrix}\textit{data-name-1}\\ \textit{literal}\end{Bmatrix}\ \underline{\text{BY}}\ \textit{data-name-2}\text{ with the result}$$

$$\underline{\text{ROUNDED}};\text{ ON }\underline{\text{SIZE}}\ \ \underline{\text{ERROR}}\ \textit{imperative-statement.}$$

In some instances, the choice can be made by default. For example, in the statement:

$$\left[;\ \underline{\text{BLOCK}}\text{ CONTAINS }[\textit{integer-1}\ \underline{\text{TO}}]\ \textit{integer-2}\begin{Bmatrix}\underline{\text{RECORDS}}\\ \text{CHARACTERS}\end{Bmatrix}\right]$$

the programmer must choose either RECORDS or CHARACTERS. If RECORDS is chosen, the word RECORDS must be written because it is a key word (indicated by the underline). However, if CHARACTERS is the choice, CHARACTERS is not a key word and the programmer may or may not write it at his discretion. When the programmer writes a BLOCK CONTAINS entry and neither RECORDS nor CHARACTERS is written, the COBOL compiler assumes that CHARACTERS was chosen and generates machine code based on this assumption.

(e) Optional functions which may be included or omitted at the user's discretion are surrounded by brackets [ ].

$$\underline{\text{MULTIPLY}}\left\{\begin{matrix}\textit{data-name-1}\\ \textit{literal}\end{matrix}\right\}\ \underline{\text{BY}}\ \textit{data-name-2}\ [\underline{\text{ROUNDED}}]$$
$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement.}]$$

(f) In some statements, certain portions may be used as many times as needed by the programmer. This repeatability is indicated by the ellipsis (...). Brackets or braces are used as delimiters to indicate the portion of the statement which is repeatable. From the foregoing, the following rule can be formed:

> Given an ellipsis (...) in a statement, scan the statement from right to left beginning at the bracket] or brace } immediately to the left of the ... until the logically matching bracket [or brace { is found; the ... applies to the words within the logically matched brackets or braces.

The following two examples illustrate the application of this rule.

Example 1:

$$\underline{\text{ALTER}}\ \textit{procedure-name-1}\ \underline{\text{TO}}\ [\underline{\text{PROCEED}}\ \underline{\text{TO}}]\ \textit{procedure-name-2}$$
$$\left[\textit{procedure-name-3}\ \underline{\text{TO}}\ [\underline{\text{PROCEED}}\ \underline{\text{TO}}]\ \textit{procedure-name-4}\right]\ .\ .\ .$$

Scanning this example from right to left, starting at the bracket immediately to the left of the ellipsis, it can be seen that the logically matching bracket is the bracket preceding procedure-name-3. Thus, the entire second line of the statement can be written as many times as the programmer chooses. The brackets surrounding PROCEED TO in both lines of the statement perform their normal function, i.e., they indicate which portion of the statement is optional.

Example 2:

$$\underline{\text{MOVE}}\left\{\begin{matrix}\textit{identifier-1}\\ \textit{literal}\end{matrix}\right\}\underline{\text{TO}}\ \{\textit{identifier-2}\}\ .\ .\ .$$

Once again, scanning from right to left, starting at the brace immediately to the left of the ellipsis, the logically matching brace is the brace immediately preceding identifier-2. The programmer may write as many different identifiers following the word TO as he chooses. The first set of braces in the statement perform their normal function; the programmer must choose either identifier-1 or literal.

The preceding illustrates the various elements of a COBOL statement. Certain language elements used in the examples (*data-name, literal, identifier, imperative-statement*) are discussed in later sections.

3. DATA DESCRIPTION

3.1. GENERAL

All data which is to be operated upon or generated by the object program must be described in the section of the source program called the *Data Division.* This division provides a standard format for the description of all data that will be input to, output from, or internally stored in the object program.

In general, data to be processed fall into the following categories:

- Data contained in files stored on some external medium and that is either brought into or sent out from object program processing.

- Data which is developed during object program processing and is placed in intermediate or working storage.

- Data which is maintained in working storage at constant value during the execution of the object program.

The exact method of writing these data descriptions, as well as the type and form of data that may appear in a COBOL program, are the subjects of this section of the manual.

3.2. CHARACTERS USED IN COBOL

The complete COBOL character set consists of 51 characters:

0,1, ... , 9

A,B, ... , Z

Blank or space

+ Plus sign

– Minus sign or hyphen

* Asterisk

/ Stroke (virgule, slash, or slant)

= Equal sign

$ Currency sign

, Comma

. Period or decimal point

; Semicolon

'' Quotation mark (substitute character is the apostrophe')

(Left parenthesis

) Right parenthesis

> Greater than symbol

< Less than symbol

The character *blank* or *space* is not an allowable character within a word (except in nonnumeric literals, explained in 3.3.1.3), 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 COBOL Reference Format (see Section 7).

3.3. TYPES OF DATA

Data to be manipulated by the main program may either be *supplied by the user* or be an *inherent part of the COBOL language* with predetermined values. The ensuing pages describe these two forms of data, their use, capabilities, restrictions, and method of describing them to the COBOL compiler.

3.3.1. User-Supplied Data

An element of data may be named or it may literally occur within the body of the program with its value interpreted as being that of the characters comprising it. For example, if the following were written:

ADD 125 TO A

the compiler would assume that ''A'' was the name of some datum and would expect an adequate description of this datum with regard to its size, form, class (alphabetic, numeric, or alphanumeric) and relationship with other data elements in the program. It would further expect that at object time an appropriate value would be supplied for this *data-name*. On the other hand, ''125'' needs no further description since it is treated literally as the integer 125 by the compiler.

The following are the rules and procedures for forming and using both named data and ''literal'' values in the source program.

3.3.1.1. Data-Names

All files, records, and elementary items used in a program must be named and described to permit referencing. These names are constructed from the standard COBOL character set and may contain any of the following characters:

0 through 9

A through Z

– (hyphen)

A data-name must contain at least one alphabetic character and may be no more than 30 characters in length.

The hyphen is used in writing compound names such as:

INVENTORY-ON-HAND

or

EMPLOYEE-RECORD

The COBOL compiler regards these as single names provided the rules for name construction are applied. They are as follows:

Hyphens may neither begin nor end a data-name. For example: the following two data-names are not legitimate:

–GROSS-PAY

and

DEPARTMENT–

The last character of a name must be followed by a space, period, comma, right parenthesis, or semicolon. When the last character is followed by any of these other than a space, the punctuation character itself must be followed by a space.

There must be no intervening space between a data-name and a following punctuation character.

An *identifier* is a data-name followed by the qualifiers, subscripts, or indexes needed to make the data-name unique.

3.3.1.2. Condition-Names

A data-name may represent not only a data entity for which various values are to be supplied during the object running of the program, but it also may represent an initially specified set of values. This data form is referred to as a *conditional variable*. Each specific value associated with the conditional variable may also be named to facilitate referencing. This name is called a *condition-name*.

For example, assume that an automobile supply dealer maintains his spare parts inventory on punched cards and that each part is coded with a specific code number in the following manner:

PART	CODE NO.
Mufflers	00764
Carburetors	92486
Batteries	39635
Tail Pipes	42666
Sparkplugs	84980

On each inventory card there might be a field called PART-TYPE with a specific code number appearing in it. To facilitate referencing and communication, names might be assigned to each code number. For example:

CODE NO.	CONDITION-NAME
00764	MUFF
92486	CARB
39635	BATT
42666	TAIL
84980	SPARK

PART-TYPE now becomes a conditional variable with a set of condition-names associated with it. These assignments are made in the data description section of the program, and thereafter, refer to the condition-names instead of the code numbers. Thus, when a card is read, the specific value of the PART-TYPE field is determined as follows:

```
IF MUFF GO TO ..........
```

This statement would generate a test of the content of the PART-TYPE field against the value 00764 and produce the same results as if the following were written:

```
IF PART-TYPE EQUALS 00764 GO TO ..........
```

The exact method of specifying condition-names, and assigning values to them, will be discussed later in this chapter. Condition-names may only be specified in a statement expressing an alternative course of action (a decision). This will become more meaningful when these statements are discussed.

3.3.1.3. Literal Data

A literal datum or a literal is a word whose value is that of the characters comprising it. That is, the quantity that is written is to be taken ''literally'' by the compiler. Literals may be alphabetic, numeric, or alphanumeric. Nonnumeric literals must be bounded by quotation marks. For example:

```
DISPLAY ''TOTAL CREDIT'' UPON PRINTER
DISPLAY ''COLUMN A1'' UPON PRINTER
```

These two lines would cause TOTAL CREDIT and COLUMN A1 to be printed out on the printer. In nonnumeric literals, it is unnecessary to include connective hyphens for the quantity to be treated as a single name as in the case of data-names. All the information contained within the quotation marks, including spaces, is treated as a single entity.

Numeric literals are created from the integers 0 through 9, a plus sign (+), a minus sign (–), and a decimal point which may appear anywhere in the word except in the rightmost position. For example:

```
ADD 128.50 TO TOTAL-DEDUCTIONS
```

This line of coding adds the quantity 128.50 to the content of a data field called TOTAL-DEDUCTIONS.

Numeric literals bounded by quotation marks are treated by the compiler as nonnumeric. That is, no arithmetic operations may be performed with or on the literal. For example:

```
ADD ''128.50'' TO TOTAL-DEDUCTIONS
```

is meaningless and would result in a diagnostic error since ''128.50'' is not equivalent to 128.50.

The following rules pertain to the formation of literal constants:

- A nonnumeric literal may not exceed 120 characters (excluding quotation marks).
- Any word (even reserved words inherent in the COBOL language) may be used as a nonnumeric literal.
- A numeric literal may not contain more than 18 digits.
- A numeric literal may contain only one sign and only one decimal point.
- The sign of a numeric literal, if present, must appear as the leftmost character. If the literal is unsigned, it is considered to be positive.
- If a numeric literal contains no decimal point, it is treated as an integer. Integral-value literals must be written in this fashion.

3.3.2. Reserved Data-Names

Certain data-names are inherent in the structure of the COBOL language and need no description. When employed, they must conform precisely to the rules established for their use. When these names are encountered in a program, the compiler can interpret them in only one way. Therefore, the programmer should make certain that he does not specify in his program a problem-oriented name that is also a reserved name. The only exception to this rule is the case of nonnumeric literals. The following types of reserved data-names are inherent in the COBOL system:

- Figurative Constants
- Special Register TALLY

3.3.2.1. Figurative Constants

Certain commonly used functions and values have been assigned fixed names and are called *figurative constants*. Unlike literals, which are actual values to be used, figurative constants are words which name values. They also differ from nonnumeric literals in that they are not enclosed in quotation marks. The figurative constants in the COBOL language are as follows:

FIGURATIVE CONSTANT	REPRESENTS
ZERO ZEROS ZEROES	Represents the value 0, or a sequence of one or more 0's depending on the context of the statement.
SPACE SPACES	Represents a sequence of one or more blank characters or spaces depending on the context of the statement.
HIGH-VALUE HIGH-VALUES	Represents one or more of the character that has the highest value in the computer's collating sequence.
LOW-VALUE LOW-VALUES	Represents one or more of the character that has the lowest value in the computer's collating sequence.
QUOTE QUOTES	Represents a sequence of quotation marks.
ALL *(any literal)*	Calls for a sequence of the specified literal. The length of the sequence is limited by the receiving field.

The singular and plural forms of the constants are equivalent and may be used interchangeably.

The following are examples of the use of figurative constants:

Assume that zeros are to be moved to a six-digit field called PART-NO. The following might be written:

```
MOVE ZERO TO PART-NO
```

PART-NO will than contain 000000. (ZEROS or ZEROES could just as easily have been used.) Similarly if it were desired to move quotation marks to PART-NO, the following might be written:

```
MOVE QUOTE TO PART-NO
```

PART-NO will then contain ’’’’’’’’’’’’. The word QUOTE cannot be used instead of quotation marks to enclose a literal. For example:

```
QUOTE NAME QUOTE
```

is not legitimate if "NAME" is intended. If it is desired to display the word NAME on the printer, the following would be written:

```
DISPLAY "NAME" UPON PRINTER
```

If the statement

```
DISPLAY QUOTE "NAME" QUOTE UPON PRINTER
```

were written, the following would be printed:

```
"NAME"
```

If it is assumed that the letter A has the highest value in a particular computer's collating sequence, and MODEL-NO is a six-character field, then the following might be written:

```
MOVE HIGH-VALUE TO MODEL-NO
```

MODEL-NO will then contain AAAAAA.

Using a six-digit field called SCALE, an example of the ALL constant is:

```
MOVE ALL "4" TO SCALE.
```

The SCALE field would then contain 444444.

3.3.2.2. Special Register

TALLY is the name of a special register whose length is equivalent to a five-decimal digit integer. Its primary use is to hold data produced by the EXAMINE verb. TALLY is provided for in each compilation and need not be described by the programmer.

The exact use of TALLY will become evident when the EXAMINE verb is considered.

3.3.3. Qualification of Data

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 and can be made unique by mentioning one or more of the higher levels of the hierarchy. The higher levels are called qualifiers when used in this way, and the process is called qualification. Sufficient qualification must be used to make a name unique, but it is not necessary to mention all levels of the hierarchy unless needed. A file-name, for example, is the highest level qualifier available for a data-name. Thus, file-names must be unique in themselves and cannot be qualified.

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 since they are logically equivalent. Names 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 the organization of two records called MASTER and NEW-MASTER, with the following partial data descriptions:

```
MASTER.....                          NEW-MASTER.....

    CURRENT-DATE.....                    CURRENT-DATE.....

        MONTH.....                           MONTH.....

        DAY.....                             DAY.....

    LAST-TRANSACTION-DATE                LAST-TRANSACTION-DATE...

        MONTH.....                           MONTH.....

        DAY.....                             DAY.....

        YEAR.....                            YEAR.....
```

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

MONTH IN CURRENT-DATE OF NEW-MASTER

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

DAY OF LAST-TRANSACTION-DATE OF MASTER

The above examples required the mention of higher levels in the organization in order to make the reference to month and day respectively unique.

This is analogous to the case where two men having the same name (John Jones), live on streets with the same name (Main Street), but in different towns. There might also be a John Jones living on 2nd Street in one town. To reference the appropriate man, you would have to specify:

John Jones of Main Street in Oshkosh or

John Jones of Main Street in Middletown or

John Jones of 2nd Street in Middletown

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

The following additional rules must be obeyed in using qualification:

- A qualifier must be of a higher level and within the same hierarchy as the name it is qualifying.
- The same name may not appear at two levels in a hierarchy so that it would appear to qualify itself.
- 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 program.
- Any data-name requiring qualification must be qualified every time it is referenced. In the absence of qualification, the COBOL compiler cannot determine the logical reference.
- 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.
- The name of a conditional variable can be used as a qualifier for any of its condition-names.
- A section-name is the highest and only qualifier for a paragraph-name. A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear in the entry. A paragraph-name need not be qualified when the reference to it is made from within the same section.
- The length of a qualified data-name is set by the specific implementation (generally, no more than 200 characters).

3.4. ORGANIZATION AND STRUCTURE OF THE DATA DIVISION

It is in the section of the program entitled DATA DIVISION that the various files, records, and elementary items to be processed are described. Storage areas set aside for holding intermediate results (or other temporarily-stored data) and constant data are also described in this division.

The Data Division consists of two sections headed as follows:

FILE SECTION.

WORKING-STORAGE SECTION.

The File Section contains two types of descriptive entries. They are:

- File description entries pertaining to each file handled by the program.
- Record description entries for each record in a given file.

The Working-Storage Section contains entries which describe areas of memory that may hold both intermediate results of processing (or other temporarily-stored data) and constant information at object time. Two types of entries may appear in the Working-Storage Section:

- Entries that describe areas which are independent and unrelated to any other area in working storage.
- Entries that describe record area or areas in which related sets of data are to reside.

The Data Division is generally organized in the following manner:

```
A   B

8   12
-------------------------------------------------------------
DATA DIVISION.

FILE SECTION.

    Description of first file

    Descriptions of all records associated with first file

    Description of second file

    Descriptions of all records associated with second file
              .
              .
              .
              .
              .
             etc.
WORKING-STORAGE SECTION.

    Descriptions of memory areas containing unrelated data items

    Descriptions of memory area containing related sets of data
```

3.5. DESCRIBING A FILE

The following pages discuss the *general procedures* for describing and defining a file in the COBOL program. These procedures may vary somewhat from system to system.

3.5.1. General

A file description is written for each file processed in the program. The information contained therein generally pertains to the physical aspects of the file. A file description consists of a mnemonic level indicator, a file-name, and a series of independent clauses which describe the physical and logical characteristics of the file.

The mnemonic level indicator, FD, is used to indicate the start of a file description. In this manner, the file description entries are distinguished from those pertaining to individual records.

The various file description clauses generally include the following information and are written in the following manner:

INFORMATION	CLAUSE
(1) The size of the physical record.	$\left[;\ \underline{\text{BLOCK}}\ \text{CONTAINS}\ [\mathit{integer\text{-}1}\ \underline{\text{TO}}]\ \mathit{integer\text{-}2}\left\{\begin{matrix}\underline{\text{RECORDS}}\\ \text{CHARACTERS}\end{matrix}\right\}\right]$
(2) The size of the data record.	$[;\ \underline{\text{RECORD}}\ \text{CONTAINS}\ [\mathit{integer\text{-}3}\ \underline{\text{TO}}]\ \mathit{integer\text{-}4}\ \text{CHARACTERS}]$
(3) The names and values of the label records contained in the file.	$;\ \underline{\text{LABEL}}\left\{\begin{matrix}\underline{\text{RECORDS}}\ \text{ARE}\\ \underline{\text{RECORD}}\ \text{IS}\end{matrix}\right\}\left\{\begin{matrix}\underline{\text{STANDARD}}\\ \underline{\text{OMITTED}}\\ \mathit{data\text{-}name\text{-}1}\end{matrix}\right\}$
(4) The value of some data contained in a label record.	$\left[;\ \underline{\text{VALUE}}\ \underline{\text{OF}}\ \mathit{data\text{-}name\text{-}2}\ \text{IS}\left\{\begin{matrix}\mathit{data\text{-}name\text{-}3}\\ \mathit{literal\text{-}1}\end{matrix}\right\}\left[,\ \mathit{data\text{-}name\text{-}4}\ \text{IS}\left\{\begin{matrix}\mathit{data\text{-}name\text{-}5}\\ \mathit{literal\text{-}2}\end{matrix}\right\}\right]\ldots\right]$
(5) The names of the data records comprising the file.	$;\ \underline{\text{DATA}}\left\{\begin{matrix}\underline{\text{RECORD}}\ \text{IS}\\ \underline{\text{RECORDS}}\ \text{ARE}\end{matrix}\right\}\mathit{data\text{-}name\text{-}5}\ [,\ \mathit{data\text{-}name\text{-}6}]\ \ldots$

The following paragraphs discuss the various clauses in detail.

3.5.2. Complete Entry Format

The organization and number of File Description clauses may vary somewhat from system to system. The programmers reference manual for the specific computer should be consulted. The following presents a general outline for clause organization:

```
 A     B
 8     12
----------------------------------------------------------------------------
 FD    file-name [; BLOCK CONTAINS [integer-1 TO] integer-2 {RECORDS   }]
                                                            {CHARACTERS}
       [; RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS]

       ; LABEL {RECORDS ARE} {STANDARD                          }
               {RECORD IS  } {OMITTED                           }
                             {data-name-1 [, data-name-2] . . . }

       [; VALUE OF data-name-3 IS {data-name-4} [, data-name-5
                                  {literal-1  }
             IS {data-name-6}] . . . ]
                {literal-2  }

       [; DATA {RECORD IS  } data-name-5 [, data-name-6] . . .].
               {RECORDS ARE}
```

The level indicator FD identifies the beginning of the file description and it must precede the file-name. It is not necessary that each clause begin a new line.

While the use of semicolons to separate clauses is optional, the last clause *must be followed by a period.* The order of appearance of the optional clauses is immaterial.

If the file description exists in a standard library of routines then the following file description may be written.

```
 A     B
 8     12
----------------------------------------
 FD    file-name COPY library-name.
```

3.5.3. File Description Clauses

The following describes the general format and rules for usage for each of the clauses that comprise the File Description Complete Entry Format.

3.5.3.1. BLOCK CONTAINS

Format:

$$\left[\ ;\ \underline{\text{BLOCK}}\ \text{CONTAINS}\ [\textit{integer-1}\ \underline{\text{TO}}]\ \textit{integer-2} \left\{\begin{matrix}\underline{\text{RECORDS}}\\ \underline{\text{CHARACTERS}}\end{matrix}\right\}\right]$$

Description:

This clause specifies the size of the *physical* record or block on tape. The physical grouping in no way affects the logic of the program; however, it may affect the amount of magnetic tape needed to store data in a tape file. With this in mind, the programmer should attempt to establish the most efficient correlation between the physical and logical record. There must be at least one record per block. Blocks may not contain partial records (i.e., records may not overlap blocks).

This clause is not required when one of the following conditions exists:

- A physical record contains only one complete logical record, e.g., one card.
- The hardware device assigned to the file has only one physical record size.
- A standard physical record size is established by the user regardless of optional sizes permitted by the equipment. If logical records of various sizes are grouped into a physical record, the end of each record must be clearly defined in the individual record description entry.

When this clause is used, the following rules apply:

- Integer-1 and integer-2 must be unsigned (positive) numeric literals.
- If only integer-2 is used, it represents the exact size of the physical record. If both integer-1 and integer-2 are used, they indicate the minimum (integer-1) and maximum (integer-2) size of the physical record.

  For example, in the statement:

  BLOCK CONTAINS 1 TO 3 RECORDS

  each block contains at least one and no more than three records. However, if "1 TO" were deleted, each block would contain three records.
- When the CHARACTERS option is selected, the physical record size is specified as the number of characters contained within the physical record regardless of the types of characters used to represent the items within the physical record.
- Whenever logical records of varying size are grouped into one physical record, the end of the logical record must be explicitly defined in the record description entry except when the user's standard technique is used.

3.5.3.2. RECORD CONTAINS

Format:

```
[; RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS ]
```

Description:

The RECORD CONTAINS clause specifies the size of the data record.

Since the size of each type of data record is completely defined within its respective Record Description entry (by the PICTURE and FILLER clauses which describe each elementary item), the RECORD CONTAINS clause is optional. When used, the following rules apply:

- Integer-1 and Integer-2 must be unsigned (positive) numeric literals.
- When integer-1 and integer-2 are both used, integer-1 refers to the number of characters in the smallest size data records, and integer-2 refers to the number of characters in the largest size data records contained in that file.
- When only integer-2 is used, it represents the exact number of characters in the data record.

  For example, in the statement:

  RECORD CONTAINS 115 TO 165 CHARACTERS.

  Each record in the file is no shorter than 115 characters and no longer than 165 characters. However, if ''115 TO'' were deleted, each record would be exactly 165 characters long.

3.5.3.3. LABEL

Format:

$$; \underline{\text{LABEL}} \left\{ \begin{array}{l} \underline{\text{RECORDS}}\ \text{ARE} \\ \underline{\text{RECORD}}\ \text{IS} \end{array} \right\} \left\{ \begin{array}{l} \underline{\text{STANDARD}} \\ \underline{\text{OMITTED}} \\ \textit{data-name-1}\ [, \textit{data-name-2}] \ldots \end{array} \right\}$$

Description:

Label records may be specified at both the beginning and end of a file. This clause permits the identification of these label records. The LABEL clause must appear for each file description.

If the user has a standard form for label records, then the STANDARD option may be used. The OMITTED option specifies that no explicit labels exist for the file or for the device to which the file is assigned. Data-name is the name of a label record which must be the subject of a Record Description entry associated with the file. The data-name must not appear in the DATA RECORDS clause of the File Description.

3.5.3.4. DATA RECORDS

Format:

$$\left[;\ \underline{\text{DATA}} \left\{ \begin{matrix} \underline{\text{RECORDS}}\ \text{ARE} \\ \underline{\text{RECORD}}\ \text{IS} \end{matrix} \right\} \textit{data-name-1}\ [,\ \textit{data-name-2}]\ldots \right]$$

Description:

This clause enables the compiler to cross-reference the File Description entry with the description of the particular logical record. Each data-name listed must be a record name. This clause is optional and serves only to document the various records in each file. These records may have different sizes and formats, and can be listed in any order.

Qualification of these data names is not permitted since they are implicitly qualified by the file-name of this entry. The data-names in this clause are the names of the records described in the Record Description entries.

Examples of this statement may be seen in the sample problem.

3.5.3.5. VALUE OF

Format:

$$\left[;\ \underline{\text{VALUE}}\ \underline{\text{OF}}\ \textit{data-name-1}\ \text{IS} \left\{ \begin{array}{l} \textit{data-name-2} \\ \textit{literal-1} \end{array} \right\} \left[,\ \textit{data-name-3}\ \text{IS} \left\{ \begin{array}{l} \textit{data-name-4} \\ \textit{literal} \end{array} \right\} \right] \ldots \right]$$

Description:

This clause specifies the value of an item in a label record. One possible use of this clause is to permit the user to enter the name of the file about to be processed in the label record. Thus, a test as to whether or not the proper file is about to be processed can be made before entering the processing stage.

Data-names 1, 3, etc., should be qualified where necessary but cannot be subscripted or indexed. Figurative constants may be used in place of the literals specified in the format. If label records are standard, then the supplied data-names must be fixed names established by the individual implementor; data-names 2, 4, etc., must be defined in the Working-Storage Section.

3.5.4. Sample Problem

In the sample problem introduced in the previous section, two input files and two output files are used: the input master inventory file, the input detail transaction file, the output (updated) master inventory file, and the output stock reorder list file.

Descriptions of the files are shown below, first in English prose and then in COBOL. The records contained in these files are described in the next section.

- Master Inventory Files – (input and output)

  The input and output master inventory files are identical in format, the latter being an updated version of the former. For this reason, their file descriptions are identical except for file-name. Each is physically stored on magnetic tape, fifty master records to a physical block with a standard label that contains a tape identifier, MSTINVTP. Both files contain a logical master record for each individual stock item in the inventory.

```
003000 DATA DIVISION.
003100 FILE SECTION.
003200 FD  OLD-MASTER-INVENTORY
003210     LABEL RECORD IS STANDARD
003250     VALUE OF ID IS 'MSTINVP'
003300     BLOCK CONTAINS 50 RECORDS
003400     DATA RECORD IS MASTER-RECORD.
                       .
                       .
                       .
                       .
                       .

006100 FD  NEW-MASTER-INVENTORY
006110     LABEL RECORD IS STANDARD
006120     BLOCK CONTAINS 50 RECORDS
006130     VALUE OF ID IS 'MSTINVP'
006140     DATA RECORD IS MASTER-RECORD.
```

- Detail Transaction File – (input)

  The detail transaction file consists of records contained on punched 80-column cards, one record per card. Label records are not used.

```
004700 FD  DETAIL-TRANSACTION-CARDS DATA RECORD IS TRANSACTI NS
004800     LABEL RECORDS OMITTED.
```

- Stock Reorder-List File – (output)

  The stock reorder-list file is printed during the object run. The record area is called Replenish Stock Item, the elements of which are as shown in the next section.

```
006500 FD  REORDER-LIST LABEL RECORD IS OMITTED
006600     DATA RECORD IS REPLENISH-STOCK-ITEM.
```

3.6. DESCRIBING A RECORD

The following pages discuss the general procedures for describing a record in the program.

3.6.1. General

Each file description is followed by the descriptions of the various records comprising that file. The record description describes all named items of data to the COBOL compiler. Like the file description, the record description comprises a number of independent clauses. These clauses may be used to describe data, or they may describe intermediate work areas. (See 3.7 for the rules and procedures regarding working-storage areas.)

Record Description clauses generally include the following information and are written in the following manner:

INFORMATION	CLAUSE
(1) The dominant usage of the data.	$\text{USAGE IS}\left\{\begin{array}{l}\underline{\text{COMPUTATIONAL}}\\ \underline{\text{COMP}}\\ \underline{\text{DISPLAY}}\end{array}\right\}$
(2) The size, class (alphabetic, numeric, alphanumeric), and editing requirements for the data.	$\left\{\begin{array}{l}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{array}\right\}\text{IS } \textit{character-string}$
(3) The justification of the data to the rightmost position in the field, or alignment by decimal point.	$\left\{\begin{array}{l}\underline{\text{JUSTIFIED}}\\ \underline{\text{JUST}}\end{array}\right\}\ \underline{\text{RIGHT}}$
(4) The positioning of items within a computer word or words.	$\left\{\begin{array}{l}\underline{\text{SYNCHRONIZED}}\\ \underline{\text{SYNC}}\end{array}\right\}\left\{\begin{array}{l}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{array}\right\}$
(5) The initial value of a working storage area or the fixed value of a condition-name or constant.	$\left\{\begin{array}{l}\underline{\text{VALUE}}\text{ IS}\\ \underline{\text{VALUES}}\text{ ARE}\end{array}\right\}\textit{literal-1}\ [\underline{\text{THRU}}\ \textit{literal-2}]$ $[,\ \textit{literal-3}\ [\underline{\text{THRU}}\ \textit{literal-4}]\ ]\ .\ .\ .$
(6) The redefinition of an area so it may be used for different data at different times.	*data-name-1* <u>REDEFINES</u> *data-name-2*
(7) Permits alternative, possibly overlapping, groupings of elementary items.	*data-name-1* <u>RENAMES</u> *data-name-2* [<u>THRU</u> *data-name-3*]

3.6.2. Complete Entry Format

Though the organization of the Record Description clauses may vary somewhat from system to system, the following presents the general method of organization:

A 8	B 12
level-number	$\left\{\begin{array}{l}\textit{data-name-1}\\ \underline{\text{FILLER}}\end{array}\right\}[;\ \underline{\text{REDEFINES}}\ \textit{data-name-2}]\left[;\ \text{USAGE IS}\left\{\begin{array}{l}\underline{\text{COMPUTATIONAL}}\\ \underline{\text{COMP}}\\ \underline{\text{DISPLAY}}\end{array}\right\}\right]$ $\left[;\left\{\begin{array}{l}\underline{\text{SYNCHRONIZED}}\\ \underline{\text{SYNC}}\end{array}\right\}\left\{\begin{array}{l}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{array}\right\}\right]\quad\left[;\left\{\begin{array}{l}\underline{\text{PIC}}\\ \underline{\text{PICTURE}}\end{array}\right\}\textit{character-string}\right]$ $\left[;\left\{\begin{array}{l}\underline{\text{JUSTIFIED}}\\ \underline{\text{JUST}}\end{array}\right\}\underline{\text{RIGHT}}\right][;\ \underline{\text{VALUE}}\ \text{IS}\ \textit{literal}]\ [;\ \underline{\text{BLANK}}\ \text{WHEN}\ \underline{\text{ZERO}}].$

When renaming an area or areas the following form is used with no additional clause specifications (see 3.6.4.9).

A 8	B 12
66	$\textit{data-name-1}\ \underline{\text{RENAMES}}\ \textit{data-name-2}\ [\underline{\text{THRU}}\ \textit{data-name-3}].$

The meaning and use of level numbers are discussed in 3.6.3.

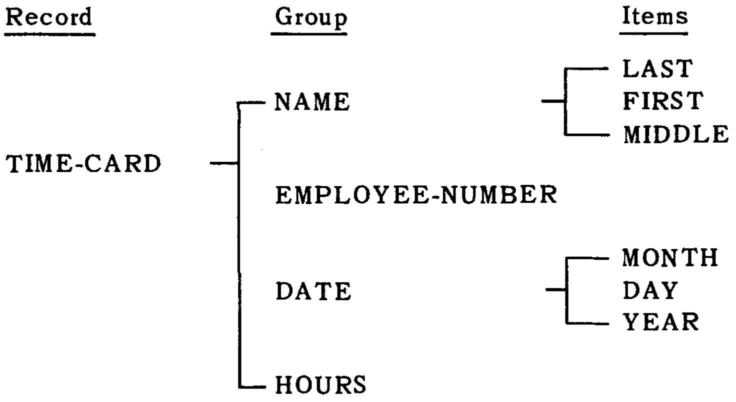
3.6.3. Record Organization

In order to indicate to the compiler the manner in which the data is organized, COBOL provides a system of *level numbers* or level indicators. Each data entry begins with a level number to indicate its position relative to other data being processed. For example, all records are assigned a level number of 01. Each subordinate entry (group items, elementary items, etc.) then takes a higher level number. Therefore, a record might be organized in the following manner:

```
01 (name and description of first record)
   02 (name and description of first group)
      03 (name and description of first item in first group)
      03 (name and description of second item in first group)
                       .
                       .
                       .
                       .
                       .
                       .
      03 (name and description of nth item in first group)
   02 (name and description of second group)
                       .
                       .
                       .
                       .
                       .
                      etc.
01 (name and description of second record)
                       .
                       .
                       .
                      etc.
```

Each group description must immediately follow the record of which it is a subset, and each item description must be listed directly after the group of which it is a part. Each datum described must be assigned a level number that is numerically greater than the larger data grouping of which it is a part.

For example, assume the record called TIME-CARD is subdivided into groups and items in the following manner:



The record organization might appear as follows:

```
01 TIME CARD
   02 NAME
      03 LAST
      03 FIRST
      03 MIDDLE
   02 EMPLOYEE-NUMBER
   02 DATE
      03 MONTH
      03 DAY
      03 YEAR
   02 HOURS
```

The organization of the data might also be written in the following manner:

```
01 TIME-CARD
   06 NAME
      10 LAST
      10 FIRST
      10 MIDDLE
   06 EMPLOYEE-NUMBER
   06 DATE
      07 MONTH
      07 DAY
      07 YEAR
   06 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 have included the various Record Description clauses.

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.

The principle rules for assigning level numbers are as follows:

- The level 01 is reserved exclusively for identifying a record.
- Level numbers range from 01 to 49.
- Any level may be an elementary item when no items are subordinate to it.
- An item is contained in the preceding group if the following conditions are met:
  - The item has been assigned a numerically higher level number.
  - The item name directly follows the group name of which it is a subset.
- Level numbers need not be assigned consecutively.

When an entry is given a numerically lower level number than the one immediately preceding it, the level number must be at least that of the preceding group. Thus, EMPLOYEE-NUMBER must be assigned at least a level 06, because that is the level number assigned to the preceding group NAME.

Certain types of data exist for which there is no true concept of level; namely, unrelated working-storage areas and condition-names.

The special level number 77 is used to describe unrelated work areas. If an entry consists of one item which is not further subdivided, and which is not a part of any larger item, then it is said to be independent. Similarly, a work area that is not composed of several parts, and is not part of a larger work area, is called an independent work area. These areas need no hierarchy of level numbers to show their relationship to other items, since they stand completely by themselves and are not a part of any hierarchy. Level number 77 entries, when used, must appear as the first entries in a description of a working storage area.

The second special level number 88 is used to identify condition-names. A condition-name is not the name of an item; it is the name of a value which a conditional variable may assume. Thus, it is given the special level number 88. Condition-names are assigned by writing an entry for the conditional variable itself, immediately followed by an entry for each condition-name to be associated with it.

Example:

```
WORKING-STORAGE SECTION.
77    Data-name-1
.
.
.
77    YTD-PAY PICTURE IS 9(7).
      88 FICA-MAX VALUE IS 0660000 THRU 9999999.
01    data-name-2
```

All 77 entries must appear before any 01 entries are written, and must be neither subordinate to another entry nor have any subordinate items. In the example shown, the 7-digit numeric item, YTD-PAY, assumes an initial value of zero. The 88-level (condition-name) entry FICA-MAX provides a simple way of determining the value of YTD-PAY with respect to the critical value 0660000.

The special level number 66 is used in connection with the RENAMES clause, as will be explained later in this section.

3.6.4. Record Description Clauses

The following describes the general format and rules for usage for each Record Description clause.

3.6.4.1. LEVEL NUMBER

Format:

$$\textit{level-number} \left\{ \begin{matrix} \textit{data-name} \\ \underline{\text{FILLER}} \end{matrix} \right\}$$

Description:

There must be a level number for each elementary item, group, or record described. Level numbers begin at 01 for records and get progressively higher for subsets of records. The highest level number is 49 (except for the special level numbers 66, 77, and 88). The data-name must conform to all rules established in 3.3.1.1.

FILLER may be used in place of data-name to indicate a portion of a record that is to contain no addressable data; e.g., a spacer between two data items that are to be printed. This specification may never appear at a 01 level. The only other clause that may be specified with FILLER in the File Section is PICTURE. In effect, FILLER may be considered a special name for data to which no data-name has been assigned, and no direct reference to FILLER may be made in the program. However, it may be referenced through the group or record of which it is a subset.

As an example, if the following were written:

```
03 LIFE-NO PICTURE IS 9(6)

03 FILLER PICTURE IS X(6)

03 NAME PICTURE IS A(24)
```

a nonaddressable 6-character field would be inserted into the record between LIFE-NO and NAME.

3.6.4.2. USAGE

Format:

$$\left[\ ;\ \text{USAGE IS} \left\{ \begin{array}{l} \underline{\text{COMPUTATIONAL}} \\ \underline{\text{COMP}} \\ \underline{\text{DISPLAY}} \end{array} \right\} \right]$$

Description:

The clause indicates to the compiler the most frequent use of the data being described and thereby dictates the format of internal representation. Data may be used in two ways:

(1) As an element in a computational procedure (COMPUTATIONAL or COMP)

(2) As an element in an operation in which the data is formatted for display (DISPLAY)

In order to improve the efficiency of the object program, it is desirable to indicate to the COBOL compiler just how data is going to be used.

The USAGE clause can be written at any level. When written at a group level it applies to each item in the group; however, the group itself is not COMPUTIONAL and cannot be used in computations. The USAGE clause of an item may not contradict the usage of a group to which the item belongs. Thus, an item cannot be described as COMPUTATIONAL when it is part of a group item specified as DISPLAY.

If the USAGE clause is not specified for an item, or for any group to which the item belongs, it is assumed to be DISPLAY.

The usage specified in this clause does not restrict in any way the operation of any verb on the data, except that COMPUTATIONAL implies numeric class. However, it does affect the way in which the data is represented (e.g., the radix may differ) inside the computer, and this will affect the efficiency of the object program.

Example:

```
03 ON-HAND-UNITS PICTURE IS 9(4) USAGE IS COMPUTATIONAL.
```

This is an instance of correct use of the USAGE clause. However, if PICTURE IS X(4) or A(4), the USAGE IS COMPUTATIONAL clause would be illegal. In the above example, the abbreviation COMP could have been used in place of COMPUTATIONAL; they are logically equivalent.

Specifying one USAGE does not preclude an item from being used according to the other. For example, COMPUTATIONAL usage might be specified for most numeric items; this does not prevent their being displayed. The opposite is also true.

A COMPUTATIONAL item may only be compared with a numeric item. A COMPUTATIONAL item may not be EXAMINEd. There are no other restrictions on the use of a COMPUTATIONAL item as an operand in any statement. That is, such an item may be DISPLAYed, MOVEd, ADDed, etc. Automatic conversion from one usage to another is supplied by the compiler.

3.6.4.3. PICTURE

Format:

$$\left[; \left\{ \begin{array}{l} \underline{\text{PICTURE}} \\ \underline{\text{PIC}} \end{array} \right\} \text{IS } \textit{character-string} \right]$$

Description:

This clause details precisely the characteristics of a particular elementary data item. It also specifies any editing that may have to be done to the data. Through this clause, the programmer may specify the class of the data item (alphabetic, numeric, alphanumeric) and its size. He may also add, delete, or alter characters thereby editing it (by means of a MOVE statement, see 4.3.7.1) into a form more useful to his own application. PIC and PICTURE are logically equivalent.

PICTURE characters fall into three general categories. They are:

Data Character Symbols – These characters indicate whether the data item is alphabetic, numeric, or alphanumeric.

Operational Symbols – These characters indicate the operational sign, the assumed decimal position, and the assumed decimal scaling position of a numeric value.

Editing Symbols – These characters indicate the editing to be done before printing an elementary item. Two types of editing symbols or characters are available.

(1) Replacement

(2) Fixed-Insertion

The replacement symbol specifies that some character in the data item (usually zero) is to be suppressed and replaced by another character (usually the symbol itself).

The fixed insertion character is inserted into the data item in addition to those characters already present.

The use of editing symbols results in two additional classes of data: alphanumeric edited and numeric edited. These classes are discussed later.

Below is a listing of all allowable PICTURE characters and the categories associated with them:

A X 9	Data Character Symbols
S V P	Operational Symbols
* Z O B , . $ + – DB CR	Editing Symbols

The maximum number of characters that can appear in a PICTURE description is thirty (30). This does not limit the number of characters in the represented area, which could be more than thirty.

An integer enclosed in parentheses can follow the symbols A X 9 P Z * $ B O – + to indicate the number of consecutive occurrences of the symbol. (That is, A(10)X(2) and AAAAAAAAAAXX are equivalent.) X(100) represents an area of 100 alphanumeric characters.

Data Character Symbols:

A represents a character position to be occupied by any of the alphabetic characters (A through Z) and the space character. For example, a five-character, alphabetic item would be represented as

AAAAA or A(5)

X represents a character position to be occupied by any character in the character set of the particular system. For example, the alphanumeric item AB1234 could be represented by any of the following:

XXXXXX

AAXXXX

A(2)X(4)

X(6)

9 represents a character position occupied by a numeric character (0 through 9). Thus, a PICTURE of 999 or 9(3) represents a three-digit field that contains a group of three numeric characters.

Operational Symbols:

The following symbols may only be used for numeric items:

S represents the position occupied by the sign of the data item. Except for the symbol P, no character can appear to the left of S in a picture.

S999 or S9(3)

represents a three-digit field that can contain a group of three numeric characters with its associated operational sign.

V represents an assumed decimal point within a data item. Thus, S9V99 or S9V9(2) represents a three character field which has an operational sign and in which there is an assumed decimal point between the third and second least significant digits. If the data item processed by the object program were −10, it would be treated as −0.10. The use of V is redundant if P is already present but V and P may appear in the same PICTURE.

P represents an assumed decimal point located outside of an item and, when used, must be either the first or the last character or characters of a PICTURE. One P is used for each assumed character position outside of the field that precedes or follows the assumed decimal point.

PPPPS9999 or P(4)S9(4)

would represent (if +8735 is substituted as the data item for which the 9999 stands)

+.00008735

Note that the P's must precede the S in this type of PICTURE.

S9999PP or S9(4)P(2)

would represent (again using 8735)

+873500.

P's cannot appear as both the first and the last characters of a PICTURE. The use of P is redundant if V is already present but P and V may appear in the same PICTURE.

Editing Symbols:

The following symbols associated with the editing functions are used when a data item is to be printed and certain of the item characters are to be suppressed or replaced, or other characters are to be inserted.

Z specifies that before the data item is printed, as many leading zeros as there are Z's are to be suppressed (replaced by a blank or space). Thus,

PICTURE	DATA ITEM	PRINTED
ZZZZ	0000	
ZZZZ	8730	8730
ZZZZ	0087	87
ZZZZ	8736	8736
ZZZ9	0087	87
ZZ99	0087	87
Z999	0087	087

* specifies that before a data item is printed, asterisks should replace leading zeros as dictated by the PICTURE clause. For example:

PICTURE	DATA ITEM	EDITED ITEM
****	0000	****
****	8730	8730
****	0087	**87
****	8736	8736
***9	0087	**87
**99	0087	**87
*999	0087	*087

An asterisk may only be preceded by a B, a zero, a currency sign, a plus sign, a minus sign, a decimal point or a comma. It can never appear in a PICTURE with Z, A, X, or S or more than one currency, minus, or plus sign.

$ may be used as either a fixed insertion character or as a replacement character. If only one $ is used in a PICTURE, then it is a fixed insertion character and will occur in the specified position within the data item when that item is printed. For example:

PICTURE	DATA ITEM	EDITED ITEM
$9999	1234	$1234
$ZZZ9	0000	$ 0
$ZZZZ	0000	

If more than one consecutive currency sign is used in the high-order end of a PICTURE, the currency sign becomes a replacement symbol. It suppresses all leading zeros as dictated by the PICTURE and inserts $ in place of the rightmost zero suppressed. However, if the value of the data is zero, then the edited item will contain spaces. For example:

PICTURE	DATA ITEM	EDITED ITEM
$$$9999	001234	$1234
$$$	000	
$$$$$	0008	$8
$$99	123	$123

The currency sign may never appear in a PICTURE with A, X, or more than one plus or minus sign.

+ or − may be used as either fixed insertion or replacement characters.

If the plus or minus sign is written as an insertion character in either the first character or last character of a PICTURE, a displayed sign (as opposed to an operational sign) is inserted into the indicated position.

When the minus sign is inserted, a minus sign will appear if the item is negative; a blank will appear in the specified position if the item is positive or unsigned. When the plus sign is used, a plus sign appears if the item is positive; a minus sign appears if the item is negative. Unsigned items are considered positive. For example:

DATA ITEM	PICTURE	EDITED ITEM
+33	−99	33
−33	99−	33−
−33	−99	−33
00	−99	00
+22	+99	+22
−22	+99	−22
20	99+	20+

If either the minus or plus sign is used as a replacement symbol, it will suppress leading zeros as dictated by the PICTURE. The rightmost zero suppressed is replaced according to the following rules:

(1) If a floating minus sign is used and the data item is negative, then a minus sign will replace the rightmost zero suppressed. If the item is positive or zero, a blank will replace it. For example:

PICTURE	DATA ITEM	EDITED ITEM
‒‒99	123	123
‒‒‒99	01$\overline{2}$	‒12
‒‒‒‒	000	

(2) If a floating plus sign is used and the data item is positive, then a plus sign will replace the rightmost zero suppressed. If the item is negative then a minus sign will replace it. For example:

PICTURE	DATA ITEM	EDITED ITEM
++99	012	+12
+++9	00$\overline{6}$	‒6
++++	000	
++99	123	+123

0 specifies that a zero is to be inserted in the item in the character position corresponding to that of the 0 in the PICTURE.

PICTURE	DATA ITEM	PRINTED
990099	8936	890036
$999.00	0736	$736.00

B specifies that a blank or space is to be inserted in the item in the character position corresponding to that of the B in the PICTURE.

PICTURE	DATA ITEM	PRINTED
99B9B9	8736	87 3 6
9BB999	8736	8 736

, specifies a character position into which a comma is to be inserted unless the preceding character has been suppressed. A comma cannot occur in a PICTURE containing any A or X characters.

PICTURE	DATA ITEM	PRINTED
99,999	87362 87000	87,362 87,000
ZZ,ZZZ	00873 20000	873 20,000

. specifies a character position into which a decimal point is to be inserted unless the succeeding character positions have been suppressed. It cannot be used in a PICTURE containing any A, X, P or V characters.

PICTURE	DATA ITEM	PRINTED
$$$999.99	0087640 (defined LEFT 2 PLACES)	$876.40

CR specifies that two character positions of the item are to contain the characters CR if the value of the data item is negative. CR can only occur as the last characters (except for P) of a PICTURE. CR cannot be used in a PICTURE containing A, X, –, +, S or DB characters.

DB specifies that two character positions of the item are to contain the characters DB if the value of the data item is negative. DB can only occur as the last characters (except for P) of a PICTURE. DB cannot be used in a PICTURE containing A, X, –, +, S or CR characters.

PICTURE	DATA ITEM	PRINTED
$$$$.99 CR	24567	$245.67
$$$$.99 CR	0013$\overline{8}$	$1.38 CR

Summarizing, the five categories of data described by a PICTURE clause are: alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited.

To define an item as alphabetic:

- Its PICTURE character-string may contain only the symbol A.
- Its contents may be any combination of the 26 letters of the alphabet and the space.

To define an item as numeric:

- Its PICTURE character-string may contain only the symbols 9, P, S, and V.
- Its contents may be a combination of the numerals 0 through 9 and may include an operational sign.

To define an item as alphanumeric:

- Its PICTURE character-string is restricted to certain combinations of the symbols A, X, and 9, but the item is treated as if the character-string contained all X's. A PICTURE character-string that consists of all X's or all 9's does not define an alphanumeric item.
- Its contents are the allowable characters in the computer's character set.

To define an item as alphanumeric edited:

- Its PICTURE character-string is restricted to certain combinations of the symbols A, X, 9, B, and 0 (zero); and must contain one of the following combinations of symbols:
  - at least one B and at least one X; or
  - at least one 0 (zero) and at least one X; or
  - at least one 0 (zero) and at least one A.
- Its contents may be any allowable character in the computer's character set.

To define an item as numeric edited:

- Its PICTURE character-string is restricted to certain combinations of the following symbols:

  B , 0 *

  P . 9 $ (or currency sign)

  V + CR

  Z – DB

- The allowable combinations are determined from the order of symbol precedence and the editing rules. A maximum of 18 digit positions may be represented in the character-string.
- The contents of the character positions must consist of numerals.

An integer enclosed in parentheses following the symbols:

A P +

X Z –

0 * $ (or currency sign)

9 B

indicates the number of consecutive occurrences of the symbol. The following symbols may appear only *once* in a given PICTURE:

S CR

V DB

.

The comma may appear more than once, but not directly adjacent to another comma.

Table 3–1 further summarizes the application of each PICTURE symbol.

PICTURE SYMBOL	REPRESENTS	CAN BE USED IN COMBINATION WITH	SPECIAL PICTURE POSITION	NOTES
A	An alphabetic character	X 9 B or 0	None	2, 3
X	An alphanumeric character	A 9 B or 0	None	1, 2, 3, 4
Q	A numeric character	Any other symbol	None	3
S	Indicates signed data	P V or 9	Leftmost	1, 5
V	Indicates position of assumed decimal point within data item	Any symbol except A or X	Must be within PICTURE. Only one V allowed	1, 4, 5
P	Indicates position of an assumed decimal point to the left or right of the data item. Each P represents one position	Any symbol except A or X	Either first or last except for S CR DB + – or $	1, 5
B	Insert space	Any symbol except S or more than one $ + or –	None	2, 4
0	Insert zero	Any symbol except S	None	4
. (period)	Insert point if following positions have not been blanked	Any symbol except A X P V S or .	None	
Z	Zero suppression; replace leading zeros with blanks	Any symbol except A S X – or more than one $ + or –	Preceded only by V . $, + – or P	4
*	Check protection; replace leading zeros with asterisks	Any symbol except Z A X S or more than one $ – or +	Preceded only by – + . V , $ or P	4
, (comma)	Insert comma unless preceding position has been blanked	Any symbol except A X S + or –	Leftmost	4

Table 3–1. PICTURE Symbols (Part 1 of 2)

PICTURE SYMBOL	REPRESENTS	CAN BE USED IN COMBINATION WITH	SPECIAL PICTURE POSITION	NOTES
$	Insert currency symbol	Any symbol except A X S + or –	Leftmost	4
or				
$$$...$	Float currency symbol	Any symbol except A X Z * or more than one + – CR or DB	Leftmost	
+	Insert correct sign	Any symbol except A X S – CR or DB	Rightmost or leftmost	4
or				
+++...+	Float correct sign	Any symbol except A X – S CR DB * Z or more than one $	Leading	
–	Insert space if value is positive, minus sign if value is negative	Any symbol except A X + S CR or DB	Rightmost or leftmost except for P	4
or				
–––...–	Float minus if value is negative	Any symbol except A X + S CR DB * Z or more than one $	None	4
CR	Insert CR if value is negative; two spaces if positive	Any symbol except A X + – S or DB	Rightmost	4
DB	Insert DB if value is negative; two spaces if positive	Any symbol except A X + – S or CR	Rightmost	4

NOTES:

(1) PICTUREs for numeric items may contain only S V P and 9
(2) PICTUREs for alphabetic items may contain only A and B
(3) PICTUREs for nonedited alphanumeric items may contain only 9 A or X
(4) PICTUREs for edited items may contain 9 V Z $ + – CR DB 0 B * and ,
(5) S V and P are not counted in the item size

Table 3–1. PICTURE Symbols
(Part 2 of 2)

3.6.4.4. JUSTIFIED

Format:

$$\left[; \begin{Bmatrix} \underline{\text{JUSTIFIED}} \\ \underline{\text{JUST}} \end{Bmatrix} \underline{\text{RIGHT}} \right]$$

Description:

This clause is used to right justify alphabetic or alphanumeric data within an area set aside to hold that particular elementary item. Standard positioning for this type of data is left justification with space fill on the right. When this statement is used the data is right justified and the unused positions are space filled. Left truncation occurs when the receiving data area is smaller than the data being moved into it. For example:

DATA ITEM	RECEIVING AREA SIZE	NORMAL POSITIONING	JUSTIFIED RIGHT POSITIONING
A9BQ7	7 characters	A9BQ7ΔΔ	ΔΔA9BQ7
MUTUAL	4 characters	MUTU	TUAL

"Δ" indicates space

This clause may not be used with numeric data since such data is either aligned by decimal point with zero fill on either end as required, or right justified in the absence of a decimal point with zero fill on the left.

JUST and JUSTIFIED are logically equivalent.

3.6.4.5. SYNCHRONIZED

Format:

$$\left[\begin{Bmatrix}\underline{\text{SYNCHRONIZED}}\\ \underline{\text{SYNC}}\end{Bmatrix} \begin{Bmatrix}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{Bmatrix}\right]$$

Description:

In many fixed word length computers, data is packed in order to conserve storage space. As a result, the data must be unpacked before it can be used thereby increasing the running time of the object program. The SYNCHRONIZED clause permits unpacked data storage by placing each elementary item in the least number of computer words required to contain it. Any unused portion of a word is filled with zeros if the item is numeric; otherwise, it is space filled. Unused portions must be accounted for in the PICTURE clause.

When SYNCHRONIZED LEFT is specified, the item starts at the leftmost boundary of the computer word. Similarly, when SYNCHRONIZED RIGHT is specified, the data is positioned so that it terminates at the rightmost boundary of the computer word. The words SYNC and SYNCHRONIZED are logically equivalent.

Items specified at the 01 level are automatically synchronized (numeric items to the right, alphabetic items to the left).

In the following examples, each computer word has a capacity of four characters, and the word boundaries are indicated by the symbol |.

Consider three adjacent data items in a word with the following PICTURE descriptions:

9(5), A(2), 9.

If no synchronization is specified they are stored in packed form, i.e.,

|9999|9AA9|

However, if synchronization is specified for any or all of the items, they may be represented in memory as follows:

	9(5)	A(2)	9	Memory
Synchronized	Right	Right	Right	|0009|9999|ΔΔAA|0009|
	Right	Left	N.S.	|0009|9999|AAΔΔ|9000|
	N.S.	Left	Right	|9999|9000|AAΔΔ|0009|
	N.S.	N.S.	Right	|9999|9AAΔ|0009|

Notes: "N.S." means "No Synchronization".
"Δ" means space.

3.6.4.6. VALUE IS

Format:

$$;\ \left\{\begin{array}{l}\underline{\text{VALUE}}\ \text{IS}\\ \underline{\text{VALUES}}\ \text{ARE}\end{array}\right\}\ \textit{literal-1}\ [\underline{\text{THRU}}\ \textit{literal-2}]\left[,\ \textit{literal-3}\ [\underline{\text{THRU}}\ \textit{literal-4}]\right]\ldots$$

Description:

This clause is used to either define the value of a condition-name or to specify the initial value of a working storage area. The literal may be any numeric or nonnumeric literal or figurative constant. The VALUE clause may be used in the Working-Storage Section to either specify the value, or range of values, of a condition-name or the value of an item to be contained therein. In the File Section, it may only be used to define the value of a condition-name and any other use of this clause in that section is for documentation purposes only. With the exception of a condition-name entry, the VALUE IS clause must not be used in an entry that either contains a REDEFINES clause or is subordinate to an entry containing a REDEFINES clause. The VALUE clause must not contradict the PICTURE clause in either length or class (numeric or nonnumeric). Further, when an item has editing symbols specified by the PICTURE clause, the literal in the VALUE clause must be nonnumeric and in edited form. A numeric literal used in this clause must have a value within the range specified by the PICTURE clause. For example, if the PICTURE of a numeric item is VPPP99, the literal specified in the VALUE clause must be within the range .00000 to .00099.

If the VALUE clause is used at a group level within the Working-Storage Section, it must not appear in the entries within the group. The group will be initialized without regard to the individual items or other groups within that group.

When used in a conditional variable, one VALUE clause must be supplied for each condition-name at an 88 level. No further entries need be specified. The form would appear in the following manner:

```
level number  data-name

              88 condition-name-1  VALUE IS  literal-1

              88 condition-name-2  VALUE IS  literal-2
                          .
                          .
                          .
              88 condition-name-n  VALUE IS  literal-n
```

Example:

```
77 PAY PICTURE IS 9(8) USAGE IS COMPUTATIONAL.
88 FICA-MAX VALUE IS 00660000 THRU ALL 9.
```

The effect of this clause is that FICA-MAX is a condition-name, dependent upon the value of PAY. If PAY is within the range of values described by the VALUE clause, the condition-name FICA-MAX becomes "true".

3.6.4.7. BLANK

Format:

[; BLANK WHEN ZERO]

Description:

This clause is used to set an item to blanks when its value is zero. If the asterisk is used as the zero suppression symbol and the BLANK WHEN ZERO clause appears in the same entry, the zero suppression editing overrides the function of the BLANK WHEN ZERO clause.

PICTURE	DATA ITEM	BLANK WHEN ZERO SPECIFIED?	EDITED ITEM
$ZZZ9	0000	No	$ 0
$ZZZ9	0000	Yes	
****	0000	No	****
****	0000	Yes	****

3.6.4.8. REDEFINES

Format:

level-number data-name-1 ; REDEFINES *data-name-2*

Description:

The REDEFINES clause allows the same computer storage area to contain different data items at different times; i.e., to "overlay" items in storage. For example, suppose a work area called MONTH-TABLE is needed in a program, and another work area, MONTH-LOOK, is used later in the same program. Normally, each area would be described separately in the Working-Storage Section, and each would occupy different portions of storage. However, if the programmer knows that MONTH-TABLE is never used when MONTH-LOOK is used, he may use the REDEFINES clause enabling both items to occupy the same physical area in storage.

The REDEFINES clause must immediately follow the entries controlled by data-name-1 (i.e., the sublevel to data-name-1). This is the only descriptive clause which must occur in a fixed place in an item description.

The level-numbers of data-name-1 and data-name-2 must be identical, and must not be 66 or 88. Also, the REDEFINES clause must not appear in 01 level entries in the File Section; implicit redefinition is provided by the DATA RECORDS clause in the File Description entry.

When an area is redefined, all descriptions of the area remain in effect. If B and C are two separate items sharing the same storage area, the procedure statements MOVE X TO B or MOVE Y TO C could be executed at any point in the program. In the first case, B would assume the value of X and take the form specified by the description of B. In the second case, the same physical area would receive Y according to the description of C. A redefinition does not cause any data to be erased and does not supersede a previous description.

Example of the REDEFINES clause:

```
03  QUANTITY.
    04  TONS PICTURE IS S9(4)V99 SYNCHRONIZED RIGHT.
    04  BASE-BOXES PICTURE IS S9(5) SYNCHRONIZED RIGHT.
    04  BASE-SYM REDEFINES BASE-BOXES.
        05  FIRST-DIGIT PICTURE IS 9.
        05  BASE-BOX-REST PICTURE IS S9(4).
```

In using the REDEFINES clause, the programmer must be extremely careful especially when either the SYNCHRONIZED clause or the SIGNED option is used. If the areas utilized by each of the data-names are not equal, program errors are likely to occur.

Except for condition-name entries, the entry describing the new storage area must not contain a VALUE clause.

3.6.4.9. RENAMES

Format:

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

Description:

The RENAMES option permits an item or items established by a record description entry to be assigned a new name. Unlike REDEFINES, this clause does not redefine existing data descriptions but merely allows data to be accessed and/or grouped under alternative names while maintaining the previously defined data description. For example, assume that a record is laid out in the following manner (this does not constitute a complete record description):

```
01 A
   02 B
      03 G
      03 H
   02 C
      03 I
      03 J
   02 D
   02 E
   02 F
```

This can be pictorially described as follows:

<table>
<tr><td colspan="7" align="center">A</td></tr>
<tr><td colspan="2" align="center">B</td><td colspan="2" align="center">C</td><td rowspan="2" align="center">D</td><td rowspan="2" align="center">E</td><td rowspan="2" align="center">F</td></tr>
<tr><td align="center">G</td><td align="center">H</td><td align="center">I</td><td align="center">J</td></tr>
</table>

The items may be renamed as follows:

```
66 K RENAMES G Thru I.
66 M RENAMES B Thru C.
66 N RENAMES E.
```

In this case, any reference to K would access items G, H, and I. Groups B and C would be accessed by a reference to M, and E would be referenced as N.

One or more RENAMES entries may be written for a record and must directly follow the last data description entry of the specific record. Data-name-1, -2, and -3 must be either elementary items or groups within the associated record and may not be the same data-name. The RENAMES clause may not be used for other 66-level entries nor can it be used for a 01, 77, or 88-level entry.

When data-name-3 is specified, data-name-1 is a group item which includes all elementary items starting with data-name-2 and concluding with data-name-3 (if data-name-3 is an elementary item) or the last item in data-name-3 (if data-name-3 is a group item).

When data-name-3 is not specified, data-name-2 can be either a group item 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. Data-name-2 must precede data-name-3 in the Record Description. Data-name-3 cannot be contained within data-name-2.

Data-name-1 cannot be used as a qualifier, and can only be qualified by names at the 01-level or FD entries.

3.6.5. Sample Problem

Data records which comprise the files are described in detail below, first using English descriptions and then the COBOL equivalent.

- Master Inventory File (input and output)

  The input and output master inventory files are each described by the same File Description; their record formats are also identical. Table 3–2 shows the breakdown of the record format, including its mnemonic name in the program.

DATA ITEM	MNEMONIC NAME	DESCRIPTION (Size and Type)	PURPOSE
Stock item number	SEQ-STOCK-NUMBER	6 alphanumeric	Unique number for each stock item to determine position in file.
Vendor number	NUMBER-MANUFACTURER	3 numeric	Code number associated with vendor.
Vendor catalogue number	MFR-CATALOG-NUMBER	10 alphanumeric	Order key specified to vendor.
Item unit description	DESCRIPTION	30 alphanumeric	Description of one order unit of the stock item.
Units on hand	ON-HAND-UNITS	4 numeric	Number of units in current stock.
Price per unit	COST-PER-UNIT	6 numeric	Wholesale price of one unit in dollars and cents.
Total wholesale value	TOTAL-WHOLESALE-VALUE	10 numeric	Product of price per unit times units on hand.
Minimum unit quantity	MIN-STOCK-UNIT-QUANTITY	4 numeric	Reorder level.

Table 3–2. Master Inventory Record Format

The COBOL version is shown below. Note that the PICTURE clause appears in each case.

```
003500 01  MASTER-RECORD.
003600     03 SEQ-STOCK-NUMBER PICTURE IS X(6).
003700     03 NUMBER-MANUFACTURER PICTURE IS 9(3).
003900     03 MFR-CATALOG-NUMBER PICTURE IS X(10).
004000     03 DESCRIPTION PICTURE IS X(30).
004100     03 ON-HAND-UNITS PICTURE IS 9(4) USAGE IS COMPUTATIONAL.
004300     03 COST-PER-UNIT PICTURE IS 9(4)V99.
004400     03 TOTAL-WHOLESALE-VALUE PICTURE IS 9(8)V99.
004500     03 MIN-STOCK-UNIT-QUANTITY PICTURE IS 9(4).
```

Table 3–3 illustrates how these records might typically appear.

STOCK NUMBER	VENDOR NUMBER	CATALOG NUMBER	ITEM UNIT DESCRIPTION	UNITS ON HAND	PRICE UNIT	TOTAL WHOLESALE	MIN. UNITS
C 0105	095	G16-264	Rifle, 264 Bolt Clip	0015	017025	0000255375	0010
C 0365	460	177-12GF	Shotgun, Dbl., FM, 12G	0033	009550	0000315150	0020
G 1931	175	CZT 146	Water skis, 1 pair	0045	002000	0000090000	0035
K 0023	984	S201	Sunglasses, men's	0300	000450	0000135000	0200

Table 3–3. Sample Master Inventory Records

■ Detail Transaction Cards

DATA ITEM	MNEMONIC NAME	DESCRIPTION	PURPOSE
Stock item number	STOCK-CONTROL-NUMBER	6 alphanumeric	Unique number for each stock item to determine position in file.
Vendor number	NO-TRANSACTOR	3 numeric	Code number associated with vendor.
Vendor catalog	ORDER-NUMBER	10 alphanumeric	Order key specified to vendor.
Item unit description	DTL-DESCRIPTION	30 alphanumeric	Description of one order unit of the stock item.
Transaction type	TYPE-TRANSACTION	1 numeric	Indicates shipment, receipt, or new stock item to be inserted in file.
Quantity of transaction	QUANTITY	4 numeric	Number of units shipped or received, or the minimum number of stock items on a new stock item.
Price per unit	UNIT-COST	6 numeric	Wholesale price of one unit in dollars and cents.

Table 3–4. Detail Transaction Record Format

The COBOL description of this record is shown below.

```
005000 01  TRANSACTIONS.
005100         05 STOCK-CONTROL-NUMBER PICTURE IS X(6).
005110         05 NO-TRANSACTOR PICTURE IS 999.
005120         05 ORDER-NUMBER PICTURE IS X(10).
005200         05 DTL-DESCRIPTION PICTURE IS X(30).
005300         05 TYPE-TRANSACTION PICTURE IS 9 USAGE IS COMPUTATIONAL.
005500         05 QUANTITY PICTURE IS 9(4).
005600         05 UNIT-COST PICTURE IS 9(4)V99.
006010         05 FILLER PICTURE IS X(20).
```

Table 3–5 shows how detail transaction records might typically appear.

STOCK NUMBER	MFR. NUMBER	CATALOG NUMBER	UNIT DESCRIPTION	TRANS CODE	QUANTITY	COST PER UNIT
C 0105	095	G16-264	Rifle, 264 Bolt Clip	1	0003	017025
C 0205	095	G88-22R	Rifle, .22 Auto	0	0030	007000
C 0205	095	G88-22R	Rifle, .22 Auto	3	0040	007000
G 1931	175	CZT 146	Water skis, 1 pair	1	0012	002000
G 1931	175	CZT 146	Water skis, 1 pair	2	0005	002000
K 0023	984	S201	Sunglasses, men's	1	0025	000450
K 0023	984	S201	Sunglasses, men's	1	0010	000450

Table 3–5. Sample Detail Transaction Records

- Reorder List File

DATA-ITEM	MNEMONIC NAME	DESCRIPTION	PURPOSE
Stock item number	STK-NUMBER-PRINT	6 alphanumeric	Printed stock item number.
Vendor number	NO-MFR	3 numeric	Printed code number associated with vendor.
New stock flag	FLAG-NEW-STOCK	1 alphabetic	Printed "N" if a new stock item; otherwise blank.
Vendor catalog number	MFR-ORDER-NUMBER	10 alphanumeric	Printed order key specified to vendor.
Item unit description	ITEM-DESCRIPTION	30 alphanumeric	Printed description of one order unit of the stock item.
Units on hand	UNITS-ON-HAND	4 numeric	Printed number of units in current stock.
Emergency reorder flag	EMERGENCY-REORDER-FLAG	3 alphabetic	Printed EEE when an emergency stock shortage; otherwise blank.
Minimum unit quantity	MIN-UNITS	4 numeric	Printed reorder level.
Price per unit	UNIT-COST	8 alphanumeric	Wholesale price of one unit of stock item in edited format.

NOTE: All of the above printed items are separated by spaces (filler) to enhance readability of the report page.

Table 3–6. Reorder List Record Format

The description of the record format would be written in COBOL as follows:

```
006700 01  REPLENISH-STOCK-ITEM.
006750     08 FILLER PICTURE IS X(5).
006800     08 STK-NUMBER-PRINT PICTURE IS X(6).
006900     08 FILLER PICTURE IS X(5).
007000     08 NO-MFR PICTURE IS 9(3).
007100     08 FILLER PICTURE IS X(7).
007250     08 FLAG-NEW-STOCK PICTURE IS A.
007300     08 FILLER PICTURE IS X(7).
007350     08 MFR-ORDER-NUMBER PICTURE IS X(10).
007400     08 FILLER PICTURE IS X(7).
007450     08 ITEM-DESCRIPTION PICTURE IS X(30).
007500     08 FILLER PICTURE IS X(5).
007550     08 UNITS-ON-HAND PICTURE IS ZZZ9.
007575     08 FILLER PICTURE IS X(5).
007600     08 EMERGENCY-REORDER-FLAG PICTURE IS X(3).
007700     08 FILLER PICTURE IS X(5).
007800     08 MIN-UNITS PICTURE IS ZZZ9.
007900     08 FILLER PICTURE IS X(5).
008000     08 UNIT-COST PICTURE IS $$$$Z.99.
008100     08 FILLER PICTURE IS X(12).
```

Table 3–7 illustrates how these records might typically appear.

STOCK NUMBER	MFR. NUMBER	NEW	MFR. CATALOG	UNIT DESCRIPTION	UNITS ON HAND	EMERG REORD	MIN UNITS	PRICE PER UNIT
C 0205	095	N	G88-22R	Rifle, .22 Auto	40		30	$70.00
G 1931	175		CZT 146	Water skis, 1 pair	28		35	$20.00
K 0023	984		S201	Sunglasses, men's	170		200	$ 4.50

Table 3–7. Sample Reorder List-Item Records

3.7. DESCRIBING WORKING STORAGE AREAS

The Working-Storage Section is used to describe areas of memory which are to contain intermediate results of processing and other temporarily-stored data at object running time. These areas may be specified in any one of three ways:

- As *single-item areas* containing data that is unrelated to any other data contained in the working-storage area.
- As *record areas* containing items of data that are interrelated and organized into records.
- As *conditional-item areas* mapped to contain conditional variables and their associated condition-names.

3.7.1. Organization and Structure

The Working-Storage Section begins with a section header followed by descriptions of both the single-item areas and the conditional-item areas. The descriptions for record areas then follow. The general format of the Working-Storage Section is as follows:

```
WORKING-STORAGE SECTION.

77  (name and description of single-item areas)
              .
              .
              .
77 (name and description of conditional variable)

   88 (condition-name-1)

   88 (condition-name-2)
          .
          .
          .
01 (name and description for record area)

   (Follows normal Record Description form.)
```

3.7.2. Single-Item Areas

These areas consist of single items which are not subdivided and are not themselves subdivisions of some other item. They are always assigned the level number 77. Each single-item area must be descr ibed in a separate data description entry consisting of the following elements:

- The level number 77.
- A data-name.
- A PICTURE clause.

Other Record Description clauses are optional and can be used to complete the description of the item, if necessary. When writing the Working-Storage Section, entries for all single items are placed before the entries describing record items.

These areas are frequently used for the temporary storage of intermediate results pending completion of a calculation. For example, suppose the programmer wishes to total several items in order to obtain an average, and he wishes to retain the total for some further calculation. In this case, the total would have to be stored temporarily. Unless it were to be used as part of a larger grouping of items, it would often be convenient to store it in some single-item working storage area.

3.7.3. Record Areas

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. Each working-storage record-name (01-level) must be unique since it cannot be qualified by a file-name. Subordinate data-names need not be unique if they can be made unique by qualification.

3.7.4. Conditional-Item Areas

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.

3.7.5. Initial Values for Working-Storage Areas

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 value may be unpredictable. All the rules for the expression of literals and figurative constants apply. The size of the literal which specifies the initial value can be equal to or less than the size specified in the PICTURE clause of the associated data entry, but it cannot be greater. When the size is less, normal rules for data positioning apply.

3.7.6. Sample Problem

The working storage data shown below are used in the sample to support the performance of various procedures.

Data Name	Description and Initial Value	Purpose
Dividend	6 numeric, zero	To store intermediate arithmetic results.
Percentage	4 numeric, zero	To hold a developed percentage.
Switch	1 numeric, zero	Assumes the value 1 when current input master is in memory.
Line number	2 numeric, zero	A line counter, to control page eject.
Replenish report heading 1		First heading line of stock reorder report, containing the word PAGE and the page number.

```
008200 WORKING-STORAGE SECTION.
008300 77  DIVIDEND PICTURE IS 9(6) VALUE IS ZERO.
008400 77  PERCENTAGE PICTURE IS 9999 VALUE IS 0000.
008500 77  SWITCH PICTURE IS 9 VALUE IS ZERO USAGE IS COMPUTATIONAL.
008610 77  LINE-NO VALUE IS 0 PICTURE IS 99.
008700 01  1-REPLENISH-REPORT-HEADING.
008800     10 FILLER PICTURE IS X(116) VALUE SPACES.
008900     10 PAGE.
009100     15 PAGEKON VALUE IS 'PAGE' PICTURE IS X(5),
009110     15 PAGE-NO PICTURE IS 999 VALUE ZERO.
009150     15 FILLER PICTURE IS X(8) VALUE SPACES.
009300 01  2-REPLENISH-REPORT-HEADING.
009350     02 ONE-THRU-SIXTY-ONE PICTURE X(61) VALUE IS '    STOCK
009400-    ' MFR      NEW       CATALOG             ITE'.
009450     02 SIXTY-TWO-THRU-132 PICTURE X(71) VALUE IS 'M - UNIT
009500-    '            UNITS   EMERG     MIN         UNIT             '.
009700 01  3-REPLENISH-REPORT-HEADING.
009750     02 ONE-THRU-71 PICTURE X(71) VALUE IS '    NUMBER     NUMBER
009800-    '  STOCK      NUMBER              DESCRIPTION'.
009900     02 SEVENTY-ONE-THRU-132 PICTURE X(61) VALUE
010000     '              ON HAND  REORD    UNITS       COST'.
```

4. PROCEDURES

4.1. GENERAL DESCRIPTION

It is in the section called the *Procedure Division* that the various instructions needed to solve a given problem are written. These instructions, though not syntactically identical to normal English, are sufficiently similar to English-language construction to permit easy communication among technical and nontechnical personnel.

The basic unit of the Procedure Division is the sentence which consists of one or more statements and/or expressions. A procedure is formed by combining one or more sentences into a paragraph and one or more paragraphs into a section. A procedure, then is a paragraph, a group of successive paragraphs, or a section within the Procedure Division. Only sections and paragraphs may be named and, therefore, are the only elements with which communication may be made.

The following pages present the various elements comprising procedures and the rules governing their use.

4.2. EXPRESSIONS

An expression in the COBOL language may be defined as a meaningful combination of data-names, literals, and operators which may be reduced to a single value. In procedural statements, two types of expressions may be specified: the *arithmetic expression* and the *conditional expression.*

4.2.1. Arithmetic Expressions

Arithmetic expressions are data-names, identifiers, or numeric literals or a series of data-names, identifiers, and literals separated by arithmetic operators which define or can be reduced to a single *numeric value.*

The format of arithmetic expressions using arithmetic operators in a COBOL procedure is as follows:

data-name

or

identifier

or

literal

or

$$\left\{\begin{matrix}\textit{data-name}\\ \textit{literal}\\ \textit{identifier}\end{matrix}\right\} \text{ operator } \left\{\begin{matrix}\textit{data-name}\\ \textit{literal}\\ \textit{identifier}\end{matrix}\right\}$$

or

$$\left\{\begin{matrix}\textit{data-name}\\ \textit{literal}\\ \textit{identifier}\end{matrix}\right\} \text{operator } \left\{\begin{matrix}\textit{data-name}\\ \textit{literal}\\ \textit{identifier}\end{matrix}\right\} \text{operator.........operator} \left\{\begin{matrix}\textit{data-name}\\ \textit{literal}\\ \textit{identifier}\end{matrix}\right\}$$

Five arithmetic operators are available for use in an arithmetic expression:

Operation	**Operator**
Addition	+
Subtraction	–
Multiplication	*
Division	/
Exponentiation	**

When arithmetic expressions are evaluated by the COBOL compiler, each term is examined and evaluated in a precise order of precedence established by the arithmetic operators. Normal precedence from high to low is as follows:

(1) Exponentiation

(2) Multiplication and Division

(3) Addition and Subtraction

The following examples illustrate the effect this ordering has on the result of an expression as opposed to a simple left to right evaluation. In these examples, let A = 4, B = 6, C = 2, D = 3, and E = 12.

	Left to Right	**COBOL Evaluation**
A + B * C ** D	8000	52
A + B * C	20	16
E – A / C	4	10

Where operators are all on the same hierarchical level (i.e., multiplication and division or addition and subtraction), evaluation occurs in left to right order.

There are instances where the rules of precedence are insufficient to represent the precise meaning of an algebraic formula. For example:

$$\frac{A + B}{A - B}$$

cannot be expressed as a COBOL expression according to the rules of precedence established above. To write it as:

$$A + B / A - B$$

is incorrect since this represents the formula

$$A + \frac{B}{A} - B$$

Parentheses are used to alter the normal rules of precedence where these rules cause ambiguities of logic in the presentation of an algebraic calculation. Parentheses are defined as having a higher order of precedence than any operator. The example above can now be accurately represented in the following manner:

$$(A + B) / (A - B)$$

Now A + B and A – B are defined as elemental factors of the division operation.

Operations within parentheses, therefore, have precedence over any other in a calculation. In the following example, let A = 4, B = 3, and C = 6:

$$A + B * A - C / 2$$

This produces 13 as a result, whereas

$$((A + B) * A - C) / 2$$

produces 11 as a result with the sequence of evaluation proceeding from the innermost parenthesized operation to the outermost, as follows:

1. A + B = 7
2. 7 * A = 28
3. 28 – C = 22
4. 22 / 2 = 11 (Result)

Double exponentiation is sometimes used in an algebraic expression, for example:

$$A^{b^{c}}$$

This may not be written as

A ** B ** C

It must be written as either

(A ** B) ** C

or as

A ** (B ** C)

whichever is intended.

In ensuing pages, it will be shown how arithmetic expressions may be used in a conditional case to determine if the expression defines a specific value.

4.2.2. Conditional Expressions

Conditional expressions are used in situations where the outcome of a test will determine the next logical steps to be performed. Like the arithmetic expression which reduces to a single numeric value, the conditional expression may be thought of as also reducing to a single value – in this case, "true" or "false". In general, truth or falsity is determined by a relational test either between a data-name and a literal, or among several data-names. For example, the expression

FICA-TO-DATE IS EQUAL TO 277.20

may or may not be true, depending upon the amount of FICA-TO-DATE accumulated by a particular employee. The outcome of this test will determine the next program steps to be executed.

Since the outcome of a relational test is used in a conditional expression to determine a course of action, and since this test is based on a comparison of characters, it is significant to discuss, at this point, just how this comparison or evaluation occurs.

4.2.2.1. Rules of Comparison

Characters are compared and evaluated on the basis of a computer collating sequence in which the characters have a specified order of magnitude. This order is "built into" the machine, and every character meaningful to the computer has its position in this ordering. The result of a comparison depends on the relative position of each character in the machine's collating sequence.

4.2.2.2. Comparison of Numeric Items

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 represents a unique value regardless of the length, sign, or implied decimal point location of an item.

For example, a comparison of a data-item which has a value of +000003 with a data-item which has a value of +03 will result in an "equal" condition. Similarly, the value of 000000 is equal to the value ±0000. Following the rules of algebra, +01 is greater than −155.

4.2.2.3. Comparison of Nonnumeric Items

For two nonnumeric items, or one numeric and one nonnumeric 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 the ordered character set. If a signed, computational item is compared with a nonnumeric item, the sign is ignored. There are two cases to consider: equal length items, and unequal length items. In a comparison of two nonnumeric items, the character in an item is compared with the corresponding character of the other item. The comparison begins with the high-order (leftmost) character of each item. If these two characters are equal, the next two are compared and so on. As soon as the unequal condition is noted, the comparison stops and the result is recorded.

(a) Items of Equal Length

If the items are of equal length, 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 low end of the item is reached. The items are determined to be EQUAL when the low-order end is reached, and no unequal pair of characters was detected.

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

(b) Items of Unequal Length

If the items are of unequal length, comparison proceeds as described above. If this process exhausts the characters of the shorter item without detection of a difference, then the shorter item is LESS THAN the longer item unless the remainder of the longer item consists solely of spaces.

4.2.2.4. The Simple Conditional Expression

A simple condition reducing to the value true of false may be expressed by any of the following:

- a relation
- a condition-name
- a sign condition
- a class condition
- a switch-status condition.

Any of the above may be used in a decision-making operation to select different paths of control in a program.

- The Relational Condition

A relational condition causes a comparison of magnitude between two quantities (or operands). Each quantity may be either an identifier, a literal, or an arithmetic expression. The general form of the relational expression is as follows:

$$\left\{\begin{array}{l} \textit{identifier-1} \\ \textit{literal-1} \\ \textit{arithmetic-expression-1} \end{array}\right\} \ \text{operator} \ \left\{\begin{array}{l} \textit{identifier-2} \\ \textit{literal-2} \\ \textit{arithmetic-expression-2} \end{array}\right\}$$

The first quantity is called the subject of the condition. The second is referred to as the object. The subject and object in a relational expression may not both be literals.

The relational operators specify the type of comparison to be made between the two quantities. A relational operator must be preceded and followed by a space. The following is a list of the relational operators:

IS [NOT] GREATER THAN

IS [NOT] >

IS [NOT] LESS THAN

IS [NOT] <

IS [NOT] EQUAL TO

IS [NOT] =

A relational expression may appear in the following manner:

AGE IS GREATER THAN 21

AGE IS NOT GREATER THAN 21

FICA-TO-DATE IS LESS THAN 277.20

REORDER-POINT IS EQUAL TO 450

GROSS IS = NET

The word NOT is provided to make the relational operator specify the exact opposite of what it would normally specify. For example:

AGE IS GREATER THAN 21

AGE IS NOT GREATER THAN 21

are exact opposites of each other. The words IS and THAN are optional and may be specified at the user's discretion without altering the meaning of the expression. For example:

FICA-TO-DATE LESS 277.20

is equivalent to

FICA-TO-DATE IS LESS THAN 277.20

or to either of the following:

FICA-TO-DATE IS LESS 277.20

FICA-TO-DATE LESS THAN 277.20

Arithmetic expressions may be operands in a relational condition. For example:

```
A + B IS EQUAL TO A + M

(A + B) ** 3 IS GREATER THAN VAR-A
```

■ The Condition-Name Condition

A condition-name is a name assigned in the Data Division to one of the values a conditional variable may assume. In a condition-name condition, the variable is tested to determine whether or not its value is equal to a value associated with a particular condition-name. If the condition-name is associated with a range of values, the variable is tested to determine if the value falls within this range. This includes both the upper and lower values. For example, the data description of a particular conditional variable might appear as follows:

```
02 MARITAL-STATUS; PICTURE IS 9;
   88 SINGLE VALUE IS 1.
   88 MARRIED VALUE IS 2.
   88 DIVORCED VALUE IS 3.
   88 WIDOWED VALUE IS 4.
```

Using an IF statement, the test to determine marital status may appear as follows:

```
IF MARRIED ..........
```

which is equivalent to

```
IF MARITAL-STATUS IS EQUAL TO 2
```

Again, as in the relational condition, NOT is used to specify the opposite of what the condition-name test would normally specify. So, for example,

```
IF NOT SINGLE ......
```

is equivalent to

```
IF MARITAL-STATUS IS NOT EQUAL TO 1
```

■ Sign Condition

This condition determines whether a numeric quantity is less than, equal to, or greater than zero. The general format of this conditional expression is as follows:

$$\left\{\begin{array}{l}\textit{identifier}\\ \textit{arithmetic-expression}\end{array}\right\} \text{ IS } [\underline{\text{NOT}}] \left\{\begin{array}{l}\underline{\text{POSITIVE}}\\ \underline{\text{NEGATIVE}}\\ \underline{\text{ZERO}}\end{array}\right\}$$

The identifier in a numeric status test must always represent a numeric value.

Any condition that can be expressed in a numeric status test may also be expressed by a relational expression. For example,

```
A + B * C IS POSITIVE
```

may also be expressed in a relational expression as

```
A + B * C IS GREATER THAN 0
```

- Class Condition

  The class condition test determines whether a quantity is purely numeric or purely alphabetic. The general format of this conditional expression is as follows:

$$\textit{identifier} \quad \text{IS } [\underline{\text{NOT}}] \left\{ \begin{array}{l} \underline{\text{NUMERIC}} \\ \underline{\text{ALPHABETIC}} \end{array} \right\}$$

  The test must be consistent with the data description of the item being tested. That is, the NUMERIC test may only be used for data which has been described as numeric and the ALPHABETIC test may be only used for data which has been described as alphabetic. Either may be used for data described as being *alphanumeric*. The usage of identifier must be defined, either explicitly or implicitly, as DISPLAY.

- The switch-status condition is used to determine whether a particular hardware switch is off or on. The implementor-name and its associated ON or OFF value must be named in the SPECIAL-NAMES paragraph of the Environment Division (see 5.3). The result of the test is true when the switch is set to the specified position corresponding to the condition-name.

4.2.2.5. The Compound Conditional Expression

A conditional expression having a single condition is referred to as a *simple conditional expression*. One containing more than one condition is referred to as a *compound conditional expression*. The various conditions in a compound expression are connected by the logical operators AND and OR and would appear in the following format:

$$\textit{simple-conditional-expression} \left\{ \begin{array}{l} \underline{\text{AND}} \text{ [NOT]} \\ \underline{\text{OR}} \text{ [NOT]} \end{array} \right\} \textit{simple-conditional-expression}$$

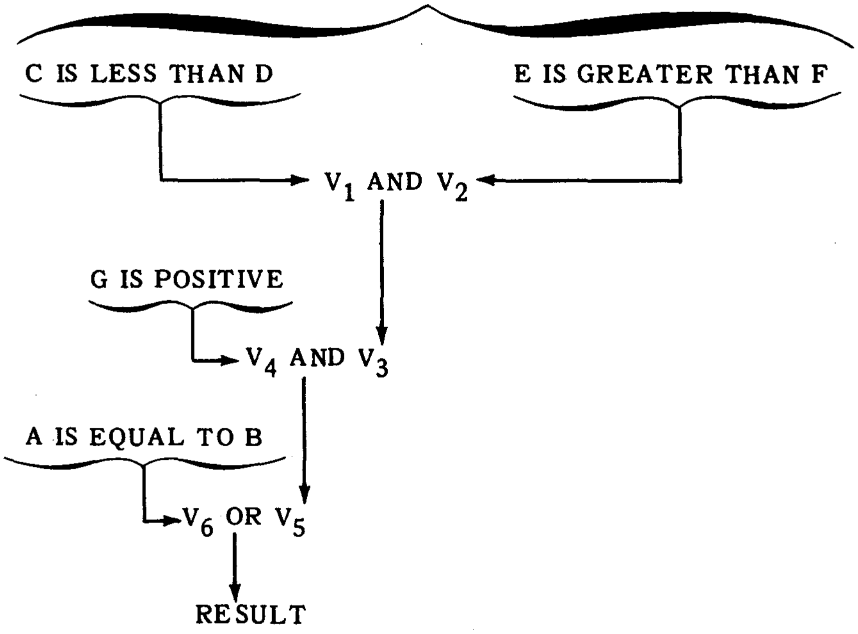
The logical operators express a logical relationship between the conditions and determine, according to rules for such operations, the truth or falsity of the entire operation. The logical determination of true or false is based upon the rules defined in the following table. Here, A and B represent simple conditions.

If A is:	and B is:	A AND B will be:	A OR B will be:	A AND NOT B will be:	A OR NOT B will be:
True	True	True	True	False	True
False	True	False	True	False	False
True	False	False	True	True	True
False	False	False	False	False	True

Two or more conditions may comprise a compound expression. Parentheses may be employed to alter the sequence of evaluation. If they are not employed, the conditions surrounding all ANDs are evaluated first, starting at the left of the expression. All ORs are evaluated next, also proceeding from left to right. For example,

A IS EQUAL TO B OR C IS LESS THAN D AND E IS GREATER THAN F AND G IS POSITIVE

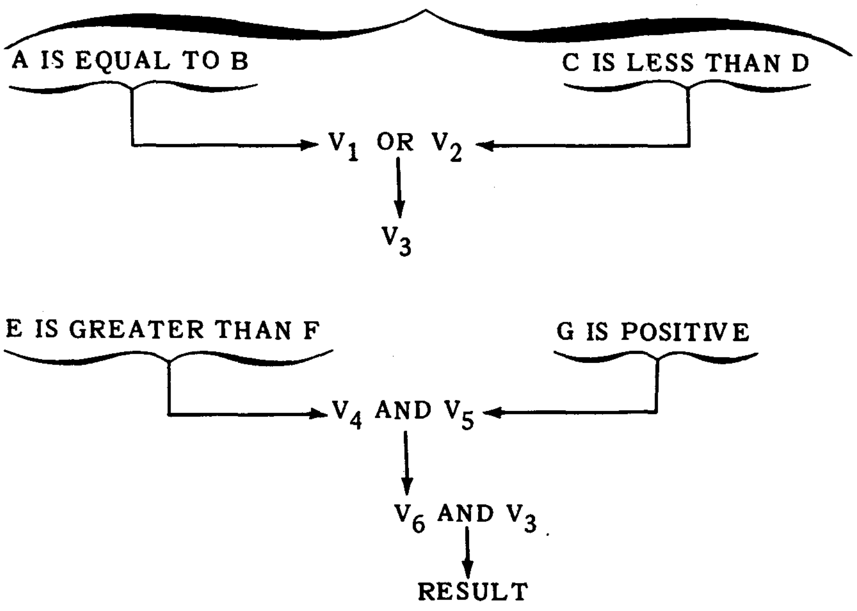
is evaluated in the following manner (here V_i represents the resultant value of an operation):



If, however, the following were written:

(A IS EQUAL TO B OR C IS LESS THAN D) AND E IS GREATER THAN F AND G IS POSITIVE

Then the evaluation would occur as follows:



4.2.2.6. Implied Subjects

In a compound conditional expression, it is possible for several consecutive conditions to have the same subject. For example,

AGE IS GREATER THAN 21 OR AGE IS LESS THAN 65

Both conditions contain the common subject AGE. Therefore, it is not necessary to state it for each consecutive condition. In other words, the subject may be implied. For example,

AGE IS GREATER THAN 21 OR LESS THAN 65

Subjects may only be implied in a series of *consecutive relational expressions,* the first of which must contain a subject, operator, and object.

4.2.2.7. Implied Operators

Operators, as well as subjects, may be implied in a compound condition consisting of a consecutive series of relational expressions. For example,

DISTRICT IS EQUAL TO 25 OR EQUAL TO 66 OR EQUAL TO 85

may be written as,

DISTRICT IS EQUAL TO 25 OR 66 OR 85

Implied operators may only be used in expressions where the subjects are also implied.

4.3. STATEMENTS AND SENTENCES

Statements form the basis functional components of COBOL procedures. Just as clauses make up sentences in normal English-language construction, so statements make up COBOL-language sentences. A sentence may contain one or more statements. There are three basic types of statements and sentences permitted in COBOL:

- Imperative
- Conditional
- Compiler-directing (see 4.7.5).

The following rules govern the construction sentences:

- Each sentence may be made up of one or more statements.
- Each sentence is terminated by a period.
- Separators may be used to enhance readability. They are optional.
- The allowable separators in a COBOL sentence are:

ƀ (space)
; (semicolon)
, (comma)

- Two contiguous separators are not permissible.
- Separators may be used in the following places:
  - Between statements.
  - Between a condition and statement-1 in a conditional sentence.
  - Between statement-1 and ELSE in a conditional statement.

4.3.1. Imperative

Imperative sentences and statements from explicit and direct commands to the computer. Imperative sentences are specified in the following manner:

verb operand
or
verb operand [;] *verb operand* [;] *verb operand* ......

For example:

ADD TEMP1 TO TEMP2.

MULTIPLY PAY-RATE BY HOURS-WORKED GIVING GROSS-PAY;
GO TO FICA-COMPUTATION.

ADD A TO B GIVING C; PERFORM 321 THRU 328.

MOVE 1050 TO REORDER-POINT.

After an imperative statement is executed, control is passed on to the next statement in sequence unless a GO TO or STOP RUN verb is present. If either is used, it must be the last statement in the sequence since control will be immediately transferred.

4.3.2. Conditional

Conditional statements and sentences are vital to any data-processing problem. In effect, they specify alternative courses of action depending upon the outcome of a test or comparison.

The format of a conditional statement is as follows:

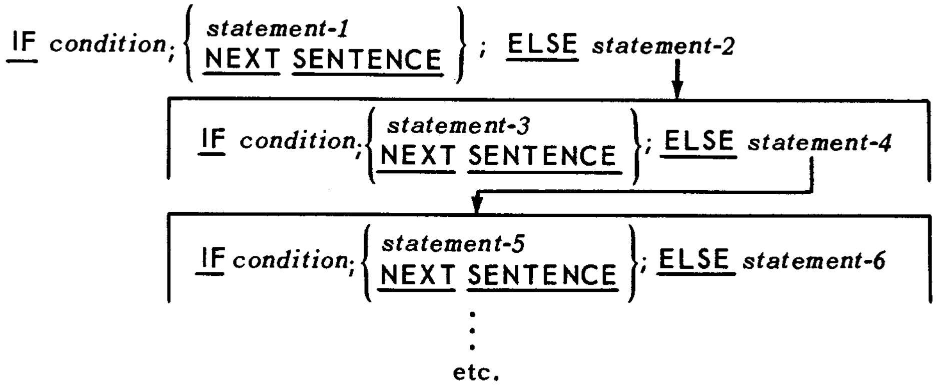
$$\underline{\text{IF}}\ \textit{condition;}\ \left\{\begin{matrix}\textit{statement-1}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{matrix}\right\}\ ;\ \underline{\text{ELSE}}\ \left\{\begin{matrix}\textit{statement-2}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{matrix}\right\}$$

Here, a conditional expression (either simple or compound) is evaluated and determined to be either true or false. If the condition is true, then statement-1 is executed and control is transferred to the next sentence. If it is false then statement-2 is executed and control is passed to the next sentence. If NEXT SENTENCE is specified instead of statement-1, control passes to the next sentence if the statement is true. If no false path is specified and the true side contains NEXT SENTENCE, for example

IF condition NEXT SENTENCE

then control passes to the next sentence regardless of whether the statement is true or false thereby having the effect of a skip-to-next-instruction operation.

Both statement-1 and statement-2 may either be imperative or conditional. If statement-2 is conditional, it may, in turn, contain another conditional statement. The conditional statement within the conditional statement is called "nesting" and may be schematically shown in the following way:



```
IF condition; { statement-1    } ; ELSE statement-2
              { NEXT SENTENCE  }
    IF condition; { statement-3   } ; ELSE statement-4
                  { NEXT SENTENCE }
    IF condition; { statement-5   } ; ELSE statement-6
                  { NEXT SENTENCE }
                  .
                  .
                  .
                 etc.
```

Some examples of conditional statements are as follows:

IF A IS GREATER THAN B; GO TO PARA-1;
ELSE GO TO PARA-2.

IF MARRIED AND AGE GREATER THAN 21; GO
TO DEP-ROUT; ELSE NEXT SENTENCE.

IF A+B**C IS POSITIVE; MOVE 1 TO D; ELSE MOVE 2 TO D.

4.4. PROCEDURE FORMATION

Procedures are formed by combining one or more sentences into a paragraph and one or more paragraphs into a section. Each paragraph or section must be preceded by a *procedure name*. These names may be either numeric, alphabetic, or alphanumeric. If numeric, leading zeros are significant (i.e., 23 is not the same as 023). All procedure names must start at position A (column 8) on the COBOL Programming Form, be a maximum of 30 characters in length, and be followed by a period.

4.5. PARAGRAPHS

One or more sentences may be combined to form a paragraph. Essentially, a paragraph expresses a single procedure to be carried through in the main program. Each program contains many such paragraphs. Each paragraph must be preceded by a procedure-name since reference may only be made to an entire paragraph and not to individual sentences contained therein. If reference is to be made to a single sentence, that sentence must be defined as a complete paragraph and must be preceded by a procedure-name.

4.6. SECTIONS

One or more paragraphs can be grouped into a section. The section is the largest unit in COBOL to which a procedure-name may be assigned. This is done by writing a procedure-name, followed by the key word SECTION, followed by a period; the remainder of the line on which it is written must be left blank. The Procedure Division need not be broken into sections if the programmer does not find it convenient.

4.7. PROCEDURAL VERBS

As in the English language, verbs specify actions to be performed. In COBOL, each verb built into the system causes a specific series of events to occur at object time. Each verb operates within the context of one or more fixed-format statements. The formats indicate the arrangement of verb and operand and indicate the particular category of procedure statement. The various verbs inherent in the COBOL language are categorized and will be described in the following manner:

Input/Output	OPEN CLOSE READ WRITE ACCEPT DISPLAY
Arithmetic	ADD SUBTRACT MULTIPLY DIVIDE COMPUTE
Data Movement and Manipulation	MOVE EXAMINE
Sequence Control	GO TO ALTER PERFORM STOP
Compiler Directing	ENTER NOTE EXIT

4.7.1. Input/Output Verbs

In any data processing application, quantities of data pass between the central storage facilities of the computer and external media such as card and tape devices. Control and coordination of the main flow of data is achieved by four input/output verbs – OPEN, CLOSE, READ, and WRITE. These verbs enable the programmer to obtain records for processing and then send the processed record to some external media. The remaining two verbs – ACCEPT and DISPLAY permit small quantities of data to be either accepted from or produced upon some external input/output device. The following pages explain the formats and uses of each of these verbs.

4.7.1.1. OPEN

Format:

$$\underline{\text{OPEN}}\ \left[\underline{\text{INPUT}}\ \left\{\textit{file-name}\ \begin{bmatrix}\underline{\text{REVERSED}}\\ \text{WITH }\underline{\text{NO}}\ \underline{\text{REWIND}}\end{bmatrix}\right\}\ldots\right]$$

$$\left[\underline{\text{OUTPUT}}\left\{\textit{file-name}\ \left[\text{WITH }\underline{\text{NO}}\ \underline{\text{REWIND}}\right]\right\}\ldots\right]$$

Description:

The function of this verb is to initiate the processing of both input and output files. Any FD entry (file) in the Data Division must be OPENed prior to the first READ or WRITE instruction directed to that particular file. This applies to the printer, card reader, and card punch as well as tape files.

The key word INPUT must be included for all input files, and the key word OUTPUT must be stated for all output files. If INPUT has been specified, the execution of an OPEN statement causes the checking of the label record if a label record has been defined in the FD entry; similarly, if OUTPUT has been specified, the OPEN statement causes the writing of the label record upon the output file.

At least one option must be specified; however, there must be no more than one instance of each option. For example, the following is permissible:

OPEN INPUT OLD-MASTER-INVENTORY, DETAIL-TRANSACTION-FILE.
OPEN INPUT OLD-MASTER-INVENTORY, OUTPUT REORDER LIST.

The following is illegal:

OPEN INPUT OLD-MASTER-INVENTORY, INPUT DETAIL-TRANSACTION-FILE.

The programmer has the facility to OPEN all input files at once or all output files at once, or to OPEN them individually as the need arises. In either case, care must be taken so that a file is not OPENed more than once unless an intervening CLOSE has been directed to the specified file.

A file may be repeatedly OPENed and CLOSEd, both for INPUT and OUTPUT, in the same program.

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.

If an input file has been designated as OPTIONAL (see 5.4.1) the file may be absent or present at object time.

When the external medium is tape, the following rules apply:

- If neither REVERSED nor NO REWIND is specified, the file is repositioned to its beginning point, i.e., the tape is rewound.
- If either REVERSED or NO REWIND is specified, no repositioning takes place. When REVERSED is specified, the file must be at its end point and subsequent READ statements cause the records to be read in reverse order. The file must be at its beginning point whenever NO REWIND is specified.

Input files OPENed in the REVERSED mode must be single-reel files. The implementor should provide a means of identifying the file to make certain that the correct tape was mounted. The method of implementing this identification is up to the user. One method, and it is probably the easiest to implement, would be to place a beginning-of-file and an end-of-file label record at the beginning and end of each file. In this way, regardless of the direction of tape movement, a label record is available for checking and identification purposes.

The REVERSED and WITH NO REWIND options are only applicable to magnetic tape files and they are meaningless when operating with card or printer files.

4.7.1.2. READ

Format:

READ *file-name* RECORD [INTO *identifier*]

; AT END *imperative-statement*

Description:

This verb makes available the next record from an input file and allows the execution of a specified imperative-statement when the end-of-file is detected. File-name must have an FD description in the Data Division.

An OPEN statement for a file must be executed before the first READ command is given. The OPEN checks the label and positions the first data record for a READ. Upon execution of the first READ, the first block is moved into the allocated area of memory and the first logical record in the file-name becomes accessible in the defined (DATA DIVISION) input area. Subsequent READ instructions advance the next logical record. For example, if MASTER has been defined as having twenty records per block, then a READ directed to MASTER will cause a record advance (positioning of the logical record) to occur twenty times for each physical movement (READ) of the tape.

When a file consists of more than one type of record, a READ delivers the next record regardless of type; stated differently, all records of a given file share the memory area. Thus, if there is more than one 01 entry in a given FD, it is the programmer's responsibility to determine which record is present at any particular instant. As an illustration, assume that HEADER and DETAIL are record descriptions of MASTER, as follows:

```
01 HEADER.
   02 IDENTITY PICTURE IS 9.
   02 DESCRIPTION PICTURE IS X(34).
         .
         .
         .
01 DETAIL.
   02 IDENTITY PICTURE IS 9.
   02 PART-CODE PICTURE IS 9(6).
   02 PART-COST PICTURE IS 9(3)V99.
         .
         .
         .
```

Since READ MASTER makes available the next record of MASTER, then IDENTITY must be interrogated to determine whether the record is a HEADER or DETAIL item. Control is then directed to a procedure that will reference only the data-name of the record type that is available. If this is not done, then a command directed to PART-CODE will, whenever a HEADER has been delivered, reference the first six positions of DESCRIPTION in HEADER instead of PART-CODE in DETAIL as intended.

When the INTO identifier option is used, the current record is read and then moved to identifier. The names of identifier and the record cannot be the same. In this instance, moving occurs according to the rules specified for the MOVE verb without the CORRESPONDING option. Under this option, "file-name RECORD" is available in the input record area as well as in the INTO area.

It is illegal to use the INTO option of the READ verb if the file that is READ has more than one record description.

Upon recognition of an end-of-reel condition, the READ causes the following operations.

- If labels are present (as defined in the FD), the standard end-of-reel label subroutine is performed.
- A tape swap occurs. If only one tape is ASSIGNed, the program will have to wait on rewind.
- If labels are present, the standard beginning reel label subroutine is executed.
- The next logical record of the file is made available.

When the logical end-of-file is reached and an attempt is made to READ the file, the imperative-statement of the AT END phrase is executed. After the imperative-statement is executed, an attempt to READ without first CLOSEing and then OPENing the file will cause an error at object time.

It is illegal to substitute any other statement for the AT END phrase.

If the file-name has been specified as OPTIONAL in the Environment Division and is not present at the object time, the imperative-statement in the AT END phrase is executed whenever a READ for that file is encountered.

4.7.1.3. WRITE

Format:

$$\underline{\text{WRITE}}\ \textit{record-name}\ [\underline{\text{FROM}}\ \textit{identifier-1}]$$

$$\left[\left\{\begin{array}{l}\underline{\text{BEFORE}}\\ \underline{\text{AFTER}}\end{array}\right\}\text{ADVANCING}\left\{\begin{array}{l}\textit{identifier-2}\ \text{LINES}\\ \textit{integer}\ \text{LINES}\\ \textit{mnemonic-name}\end{array}\right\}\right]$$

Description:

The WRITE verb releases a unit record to an output file, and allows vertical positioning if the output medium is an online printer.

The area to be written (record-name) must be defined in the Data Division at the 01 level. The file associated with record-name must also be defined by an FD entry in the Data Division. The file must be OPENed prior to the execution of the first WRITE for that file.

When the WRITE is executed, record-name is released for the output file, and thus, is no longer available.

The FROM identifier-1 option is similar to the INTO data-name option of the READ verb. Use of this option, in essence, converts the WRITE to a MOVE and WRITE. If the format of identifier-1 differs from that of record-name, the data is moved in accordance with the rules for the MOVE verb without the CORRESPONDING option. While the information in record-name is no longer available, the data in identifier-1 continues to be accessible. The names of identifier-1 and record-name cannot be the same.

The ADVANCING option allows control of the vertical positioning of each record on the printed page. The following rules are pertinent to the option:

- When identifier-2 is used, it must have a positive integral value. The compiler inserts a mechanism into the object program which positions the printer page according to the current value of identifier-2.

- When integer is used, it must be a positive integral literal. The compiler inserts a mechanism into the object program which advances the printer page *integer* lines.

- When mnemonic-name is used, it is associated with a particular feature specified by the user and it must be defined in the SPECIAL-NAME paragraph of the Environment Division.

- If mnemonic-name is specified, the printer is advanced according to the rules specified by the user.

- BEFORE and AFTER ADVANCING can result in over printing. A matrix of the print-space operation follows:

	Present Write		
Previous Write	W	WAA n	WBA
W	LINE-SPACING THEN PRINT	SPACE n THEN PRINT	OVER PRINT
WAA	LINE-SPACING THEN PRINT	SPACE n THEN PRINT	OVER PRINT
WBA m	SPACE THEN PRINT	SPACE m + n THEN PRINT	SPACE THEN PRINT

LEGEND:

W	= WRITE
WAA	= WRITE AFTER ADVANCING
WBA	= WRITE BEFORE ADVANCING
m	= Space information of data-name-2 or integer
n	= Same as m

After recognition of end-of-reel, the WRITE performs the following operations:

- The standard end-of-reel label subroutine if labels are specified in the FD of the file.
- A tape swap.
- The standard beginning-of-reel subroutine if labels are specified in the FD of the file.

4.7.1.4. CLOSE

Format:

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

    [, file-name-2 [REEL] [WITH {NO REWIND}]] ...
                                {LOCK     }
```

Description:

The CLOSE verb terminates the processing of one or more input or one or more output files or reels and provides optional rewinding and/or locking. Each file-name refers to an FD description in the Data Division and an OPEN statement must be executed prior to the CLOSE statement.

The CLOSE file-name option, as applied to the entire file rather than to individual reels, initiates the final closing conventions for the file and releases the data area. A file may be CLOSEd once, but not more than once, for each time the file is OPENed.

For an output file, the final closing conventions such as block padding, etc., for the file are performed and the data area is released. Furthermore, for either an input or an output file:

- If neither LOCK nor NO REWIND is specified, the current reel of the file is rewound and all other reels belonging to the file are rewound. However, this rule does not apply to those reels controlled by a prior CLOSE REEL entry.

- If the NO REWIND option is specified, the current reel of the file remains in whatever position it is in at the time the CLOSE is given.

- The REEL, NO REWIND, and LOCK options are only applicable to magnetic tape files and they are meaningless when operating with card or printer files.

- If the LOCK option is specified, all reels belonging to the file are rewound with interlock except for those reels controlled by a prior CLOSE REEL.

The CLOSE file-name REEL option may be used for input or output files. The LOCK option may be used and the current reel will be rewound with interlock. The necessary processing is performed.

When a CLOSE REEL is given, the locking and rewinding options of CLOSE REEL, if used, take precedence for the current reel and only the current reel, regardless of the options associated with a CLOSE of file. When a CLOSE file-name is given, its options are executed wherever possible, for all mounted reels of the file except for those reels which may have been closed by a CLOSE REEL whose locking and rewinding options differ from those of the CLOSE file-name.

If the file has been specified as OPTIONAL (see 5.4.1) the standard end-of-file processing is not performed when this file is not present.

For multiple reel files, the opening and closing of individual reels is automatic. However, the programmer must close the file when processing is to be terminated. A CLOSE file-name-1 should be executed for each file that was OPENed.

The following example illustrates the use of the LOCK option in terminating a run.

```
CLOSE EDITED-SHIPMENTS WITH LOCK, COST-OF-SALES-RATES
WITH LOCK, COSTED-SHIPMENTS WITH LOCK, NOT-COSTED-
ITEMS WITH LOCK, LOSS-ITEMS WITH LOCK, ERROR-LISTING
WITH LOCK.
```

4.7.1.5. ACCEPT

Format:

<u>ACCEPT</u> *identifier* [<u>FROM</u> *mnemonic-name*]

Description:

This verb is used to read low-volume data from the specified hardware device. The hardware device associated with a mnemonic-name must be specified in the SPECIAL-NAMES paragraph of the Environment Division when the FROM option is employed.

In many cases, a standard hardware device is used for a particular implementation of COBOL, thereby making the FROM clause unnecessary. Also, a maximum sizc for the data represented by identifier will be set. The individual supplement manuals for a particular computer system should be consulted for this information. If the data ACCEPTed is less than the maximum size for the particular system, it appears in the leftmost positions of the input area with zero fill if the data is numeric and space fill if alphabetic or alphanumeric.

For example:

```
DISPLAY "FURNISH DATE" UPON CONSOLE.
ACCEPT PRESENT-DATE FROM CONSOLE.
```

Previously, the console typewriter was designated as CONSOLE in the SPECIAL-NAMES paragraph of the Environment Division. When the DISPLAY statement is executed, FURNISH DATE appears on the console typewriter. Control passes to the ACCEPT statement and the program waits for the operator to type in the current date, after which the data accepted is stored in location PRESENT-DATE. Control then passes to the statement following the ACCEPT statement.

4.7.1.6. DISPLAY

Format:

$$\underline{\text{DISPLAY}} \left\{\begin{matrix} \textit{literal-1} \\ \textit{identifier-1} \end{matrix}\right\} \left[, \left\{\begin{matrix} \textit{literal-2} \\ \textit{identifier-2} \end{matrix}\right\} \right] \ldots$$

[UPON *mnemonic-name*]

Description:

This verb displays low-volume data on an output device.

The hardware device associated with a mnemonic-name must be specified in the SPECIAL-NAMES paragraph of the Environment Division when the UPON option is employed.

A specific peripheral unit may be designated as the standard display device thereby making the UPON clause unnecessary. Maximum length for DISPLAYed data is set by the implementor. Rules for positioning are the same as for the ACCEPT verb.

Literals and identifiers may be used in combination in a DISPLAY statement. Any figurative constants, except ALL, may be used.

For example:

```
DISPLAY "TOTAL AMOUNT IS" TOTAL-AMOUNT
```

Assume TOTAL-AMOUNT has a value of 4800 at the time the DISPLAY statement is executed. The information that appears on the display device is as follows:

```
TOTAL AMOUNT IS 4800.
```

Since "TOTAL AMOUNT IS" is a nonnumeric literal, it is displayed as is.

4.7.2. Arithmetic Verbs

The arithmetic verbs permit basic calculations to be performed on the data. Four verbs are provided in COBOL corresponding to the four basic arithmetic operations: ADD, SUBTRACT, MULTIPLY, and DIVIDE. In addition, a fifth verb, COMPUTE, allows the programmer to effect arithmetic calculations through the use of arithmetic expressions.

The following general rules pertain to the arithmetic verbs:

(a) All identifiers used in arithmetic statements must represent numeric data defined in the Data Division. The results are unpredictable if the identifiers contain other than numeric data at object time.

(b) All literals used in arithmetic statements must be numeric.

(c) The maximum size of any operand (identifier or literal), intermediate result, or receiving item is 18 digits.

(d) The formats (PICTURE) of multiple operands in an arithmetic statement may differ from each other. Decimal point alignment is supplied automatically throughout computations. Conversion of items with unlike usage is automatic.

(e) The format of any data item involved in computations (e.g., addends, subtrahends, multipliers,etc.) cannot contain editing symbols. The compiler will indicate an error by an appropriate message when the fields involved are defined in such a way that they would contain editing symbols. Operational signs and implied decimal points are not considered editing symbols. The identifiers in the GIVING option represents data items which must not enter into computations if they contain editing symbols.

(f) If the number of fractional places in a computed result (sum, difference, product, or quotient) exceeds the number of fractional places in the format of the identifier associated with the result (i.e., the identifier that is to take on the value of the result), truncation occurs unless the ROUNDED option has been used.

Truncation is the dropping of excess digits; it is always determined by the PICTURE of the identifier associated with the result. When ROUNDED is specified, however, the least significant digit specified by the format of the result is increased by 1 whenever the most significant digit of the excess is greater than or equal to 5. For example, with a receiving item PICTURE of 9(4)V9, the value 8250V96 becomes 8251V0 if the ROUNDED option is specified, and 8250V9 when ROUNDED is not used.

(g) Whenever the number of integral places (i.e., those to the left of the decimal point) in the calculated result exceeds the number of the integral places associated with the resultant identifier, 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.

Use of ON SIZE ERROR must be carefully controlled. This clause does not substitute for proper investigation and record design.

(1) The testing for the size error condition occurs only when the ON SIZE ERROR option is specified in the verb format. In the event that ON SIZE ERROR is not specified, and a size error condition arises, the results are unpredictable.

(2) If the ON SIZE ERROR option has been specified, and a size error condition arises, then the value of the resultant identifier is not altered. The imperative-statement associated with the ON SIZE ERROR option is executed after the last resultant identifier is considered.

4.7.2.1. ADD

Format:

Option 1:

$$\underline{\text{ADD}}\ \begin{Bmatrix} \textit{identifier-1} \\ \textit{literal-1} \end{Bmatrix}\ \left[,\begin{Bmatrix} \textit{identifier-2} \\ \textit{literal-2} \end{Bmatrix} \right] \ldots$$

$$\underline{\text{TO}}\ \textit{identifier-m}\ [\underline{\text{ROUNDED}}]\ [,\ \textit{identifier-n}\ [\underline{\text{ROUNDED}}]]\ \ldots$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Option 2:

$$\underline{\text{ADD}}\ \begin{Bmatrix} \textit{identifier-1} \\ \textit{literal-1} \end{Bmatrix}\ ,\begin{Bmatrix} \textit{identifier-2} \\ \textit{literal-2} \end{Bmatrix}\ \left[,\begin{Bmatrix} \textit{identifier-3} \\ \textit{literal-3} \end{Bmatrix} \right] \ldots$$

$$\underline{\text{GIVING}}\ \textit{identifier-m}\ [\underline{\text{ROUNDED}}]$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Option 3:

$$\underline{\text{ADD}}\ \begin{Bmatrix} \underline{\text{CORRESPONDING}} \\ \underline{\text{CORR}} \end{Bmatrix}\ \textit{identifier-1}\ \underline{\text{TO}}\ \textit{identifier-2}\ [\underline{\text{ROUNDED}}]$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Description:

This verb permits the addition of two or more data items, storing the result in the last specified data item of the statement. This last data item may not be a literal.

When the TO option is used, the values of the data-names and literals to the left of the word TO are added; the resulting sum is then added to the data-name to the right of the word TO. The results of the addition are stored in the identifier(s) that follows the word TO. For example, ADD A, B, C TO A is equivalent to ADD A, B, C GIVING TEMP: ADD TEMP TO A.

When the GIVING option is used, the sum of the values of the identifiers or literals preceding the word GIVING is placed in identifier-m. Since identifier-m is a receiving item, it may be an edited item.

If the CORRESPONDING option is used, multiple ADD operations are performed. The individual operations are each equivalent to ADD identifiers of numeric elementary items subordinate in hierarchy to the identifier-1 and identifier-2 written in the ADD CORRESPONDING statement. Identifiers match if they and all their possible qualifiers up to, but not including identifier-1 and identifier-2 are the same. In determining which identifiers are CORRESPONDING, any identifier subordinate to identifier-1 or identifier-2 which have REDEFINES clauses are ignored, as well as any identifiers which are subordinate to the REDEFINEd identifiers. This restriction does not preclude identifier-1 or identifier-2 themselves from having REDEFINES clauses or from being subordinate to identifiers with REDEFINES clauses. Identifier-1 or identifier-2 cannot have a level number of 66, 77, or 88. Each individual operation defined by an ADD CORRESPONDING statement is subject to the rules which apply to the ADD verb.

To illustrate an Option 3 addition, assume that the following record descriptions have been written in the Data Division of an accounting program:

```
01 MONTHLY-SALARY-EXPENSE        01 TOTAL-SALARY-EXPENSE
   02 PLANT                         02 PLANT
   02 OFFICE                        02 OFFICE
   02 SALARY                        02 SALARY
   02 EXECUTIVE                     02 EXECUTIVE
```

To update the TOTAL-SALARY-EXPENSE record, the programmer would write:

```
ADD CORR MONTHLY-SALARY-EXPENSE TO TOTAL-SALARY-EXPENSE.
```

When this instruction is executed, the value in each field of MONTHLY-SALARY-EXPENSE would be added, respectively, to the value in the similarly named fields of TOTAL-SALARY-EXPENSE. CORR and CORRESPONDING are logically equivalent.

Further examples:

Statement	Result Field PICTURE IS:	Calculation
ADD A, B TO C.	9999	A+B+C stored in C as xxxx
ADD A, B, C TO D.	$9999.99	Error – operand may not contain editing symbols except with GIVING option.
ADD A, B, C TO D.	S9999V99	A+B+C+D stored in D as + xxxxVxx
ADD A, B, C GIVING D.	$9999.99	A+B+C stored in D as $xxxx.xx
ADD 1, 5, C TO 7.		Error – result cannot be stored in literal.
ADD A, 14 TO C ROUNDED.	99999	A+14+C stored in C as $x_1x_2x_3x_4x_5$; rounded if $x_6 \geq 5$
ADD A, B, 43.6 GIVING D ON SIZE ERROR GO TO O–FLOW.	99V99	A+B+43.6 stored in D; if integer result is greater than 2 digits, SIZE ERROR occurs.

NOTE: x's show result format.

4.7.2.2. SUBTRACT

Format:

Option 1:

$$\underline{\text{SUBTRACT}} \begin{Bmatrix} \textit{literal-1} \\ \textit{identifier-1} \end{Bmatrix} \left[, \begin{Bmatrix} \textit{literal-2} \\ \textit{identifier-2} \end{Bmatrix} \right] \ldots$$

$$\underline{\text{FROM}}\ \textit{identifier-m}\ [\underline{\text{ROUNDED}}] \left[, \textit{identifier-n}\ [\underline{\text{ROUNDED}}] \right] \ldots$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Option 2:

$$\underline{\text{SUBTRACT}} \begin{Bmatrix} \textit{literal-1} \\ \textit{identifier-1} \end{Bmatrix} \left[, \begin{Bmatrix} \textit{literal-2} \\ \textit{identifier-2} \end{Bmatrix} \right] \ldots\ \underline{\text{FROM}} \begin{Bmatrix} \textit{literal-m} \\ \textit{identifier-m} \end{Bmatrix}$$

$$\underline{\text{GIVING}}\ \textit{identifier-n}\ [\underline{\text{ROUNDED}}]$$

$$[\ ;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Option 3:

$$\underline{\text{SUBTRACT}} \begin{Bmatrix} \underline{\text{CORRESPONDING}} \\ \underline{\text{CORR}} \end{Bmatrix} \textit{identifier-1}\ \underline{\text{FROM}}\ \textit{identifier-2}\ [\underline{\text{ROUNDED}}]$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Description:

This verb causes one numeric data item or the sum of two or more numeric data items, to be subtracted from a specified numeric data item. The result is stored in the last specified data item of the statement. This last item must not be a literal.

All previously stated rules regarding the ON SIZE ERROR option, the ROUNDED option, the GIVING option, the CORRESPONDING option, truncation, the REDEFINEd caution, and the editing of results apply to the SUBTRACT verb.

The individual SUBTRACT operations defined by the CORRESPONDING option are each equivalent to SUBTRACT identifier-1 FROM identifier-2.

Examples:

Statement	Result Field PICTURE IS:	Calculation
SUBTRACT 16, A, B FROM D.	999	D – (16 + A + B) stored in D as xxx.
SUBTRACT A, B FROM D.	$$99.99	Error – operand may not contain edit symbols unless GIVING option is used.
SUBTRACT A, B FROM 126.		Error – result cannot be stored in literal.
SUBTRACT A, B FROM 126 GIVING C.	999	126 – (A+B) stored in C as xxx.

NOTE: x's show result format.

4.7.2.3. MULTIPLY

Format:

Option 1:

$\underline{\text{MULTIPLY}} \left\{ \begin{matrix} \textit{identifier-1} \\ \textit{literal} \end{matrix} \right\} \underline{\text{BY}} \; \textit{identifier-2} \; [\underline{\text{ROUNDED}}]$

$[\text{; ON } \underline{\text{SIZE}} \; \underline{\text{ERROR}} \; \textit{imperative-statement}]$

Option 2:

$\underline{\text{MULTIPLY}} \left\{ \begin{matrix} \textit{identifier-1} \\ \textit{literal-1} \end{matrix} \right\} \underline{\text{BY}} \left\{ \begin{matrix} \textit{identifier-2} \\ \textit{literal-2} \end{matrix} \right\}$

$\underline{\text{GIVING}} \; \textit{identifier-3} \; [\underline{\text{ROUNDED}}]$

$[\text{; ON } \underline{\text{SIZE}} \; \underline{\text{ERROR}} \; \textit{imperative-statement}]$

Description:

This verb multiplies two numeric data items and stores the resulting product in the last data item specified in the statement.

When Option 1 is used, the value of identifier-1 or literal is multiplied by the value of identifier-2. The value of identifier-2 is replaced by the product of the multiplication.

When Option 2 is used, identifier-3 contains the product of the multiplication.

All previously stated rules regarding the ON SIZE ERROR option, the ROUNDED option, the GIVING option, truncation, the REDEFINEd caution, and the editing of the results apply to the MULTIPLY verb. For MULTIPLY, the ON SIZE ERROR option rules apply to the intermediate results as well as the final results of the operation.

Examples:

Statement	Result Field PICTURE IS:	Calculation
MULTIPLY A BY B.	999	A x B stored in B as xxx.
MULTIPLY HOURS BY 100.		Error – result cannot be stored in literal.
MULTIPLY HOURS BY 100 GIVING GROSS.	9999	HOURS x 100 stored in GROSS as xxxx.
MULTIPLY 12 BY B.	$$9.99	Error – no editing without GIVING option.

NOTE: x's show result format.

4.7.2.4. DIVIDE

Format:

Option 1:

$$\underline{\text{DIVIDE}} \begin{Bmatrix} \textit{identifier-1} \\ \textit{literal} \end{Bmatrix} \ \underline{\text{INTO}} \ \textit{identifier-2} \ [\underline{\text{ROUNDED}}]$$

[; ON SIZE ERROR *imperative-statement*]

Option 2:

$$\underline{\text{DIVIDE}} \begin{Bmatrix} \textit{identifier-1} \\ \textit{literal-1} \end{Bmatrix} \underline{\text{INTO}} \begin{Bmatrix} \textit{identifier-2} \\ \textit{literal-2} \end{Bmatrix} \ \underline{\text{GIVING}} \ \textit{identifier-3} \ [\underline{\text{ROUNDED}}]$$

[; ON SIZE ERROR *imperative-statement*]

Option 3:

$$\underline{\text{DIVIDE}} \begin{Bmatrix} \textit{identifier-1} \\ \textit{literal-1} \end{Bmatrix} \ \underline{\text{BY}} \ \begin{Bmatrix} \textit{identifier-2} \\ \textit{literal-2} \end{Bmatrix} \ \underline{\text{GIVING}} \ \textit{identifier-3} \ [\underline{\text{ROUNDED}}]$$

[; ON SIZE ERROR *imperative-statement*]

Option 4:

$$\underline{\text{DIVIDE}} \begin{Bmatrix} \textit{identifier-1} \\ \textit{literal-1} \end{Bmatrix} \underline{\text{INTO}} \begin{Bmatrix} \textit{identifier-2} \\ \textit{literal-2} \end{Bmatrix} \ \underline{\text{GIVING}} \ \textit{identifier-3}$$

[ROUNDED] REMAINDER *identifier-4*

[; ON SIZE ERROR *imperative-statement*]

Option 5:

$$\underline{\text{DIVIDE}} \begin{Bmatrix} \textit{identifier-1} \\ \textit{literal-1} \end{Bmatrix} \ \underline{\text{BY}} \ \begin{Bmatrix} \textit{identifier-2} \\ \textit{literal-2} \end{Bmatrix} \ \underline{\text{GIVING}} \ \textit{identifier-3}$$

[ROUNDED] REMAINDER *identifier-4*

[; ON SIZE ERROR *imperative-statement*]

Description:

This verb divides one numeric data item by another and stores the resulting quotient in the last data item specified in the statement.

In Option 1, identifier-1 is the divisor and identifier-2 the dividend. The value of identifier-2 is replaced by the value of the quotient.

In Option 2, identifier-1 is the divisor, and identifier-2 is the dividend. The quotient resulting from the division is identifier-3. This data name may be an edited item.

In Option 3, identifier-2 is the divisor, and identifier-1 is the dividend. The quotient is placed in identifier-3. This data may be an edited item.

Options 4 and 5 are used when a remainder from the division operation is desired, namely identifier-4. A remainder in COBOL is defined as a result of subtracting the product of the quotient and the division from the dividend. If the ROUNDED option is specified, the quotient is rounded after the remainder is determined.

The identifiers used must reference numeric elementary items whose descriptions appear in the Data Division of the program.

All previously stated rules regarding the ON SIZE ERROR option, the ROUNDED option, the GIVING option, the REDEFINEd caution, truncation, and the editing of results, apply to the DIVIDE verb. For DIVIDE, the ON SIZE ERROR option rules apply to the intermediate results as well as the final results of the operation.

An error is indicated at compilation time if the data description for either identifier-1 or identifier-2 specifies the presence of editing symbols. Division by zero results in a size error.

Examples:

Statement	**Result Field PICTURE IS:**	**Calculation**
DIVIDE A INTO B .	9(4)V9(2)	B ÷ A stored in B as xxxx.xx
DIVIDE A INTO B.	$$99.99	Error – editing not permitted except with GIVING option.
DIVIDE A INTO B GIVING C.	S999V99	B ÷ A stored in C as +xxxVxx
DIVIDE A BY B GIVING C.	9(5)	A ÷ B stored in C as xxxxx

NOTE: x's show result format.

4.7.2.5. COMPUTE

Format:

$$\underline{\text{COMPUTE}}\ \textit{identifier-1}\ [\underline{\text{ROUNDED}}]\ \underline{=}\ \left\{\begin{array}{l}\textit{identifier-2}\\ \textit{literal}\\ \textit{arithmetic-expression}\end{array}\right\}$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Description:

This verb causes one or more numeric data items (defined and established in the Data Division) to assume a new value derived from either a named or literal numeric data item or an arithmetic expression.

For example:

```
COMPUTE A = (B+C)/D–E)

COMPUTE FICA-ACCUM = TOT-FICA

COMPUTE NET-PAY = GROSS-DED
```

Only identifier-1 may contain editing symbols. Identifier-2 must be an elementary numeric item.

4.7.3. Data Movement and Manipulation Verbs

Several COBOL verbs have the ability to move or manipulate data in some manner. However, this aspect is of secondary importance and only incidental to the specific objective of the verb. For example, the arithmetic verbs may involve some data movement and/or manipulation. This, however, is secondary to their main function of effecting an arithmetic calculation. Two verbs are provided in COBOL for the specific purpose of moving or manipulating data. They are the MOVE and EXAMINE verbs. The primary purpose of the MOVE verb is to transmit data from one area of computer storage to another. The EXAMINE verb, however, causes data to be examined and moved only when certain factors are present.

4.7.3.1. MOVE

Format:

Option 1:

$$\underline{\text{MOVE}} \left\{\begin{matrix} \textit{identifier-1} \\ \textit{literal} \end{matrix}\right\} \underline{\text{TO}} \left\{\textit{identifier-2}\right\} \quad \ldots$$

Option 2:

$$\underline{\text{MOVE}} \left\{\begin{matrix} \underline{\text{CORRESPONDING}} \\ \underline{\text{CORR}} \end{matrix}\right\} \textit{identifier-1} \ \underline{\text{TO}} \ \textit{identifier-2}$$

Description:

The MOVE verb transfers information from one data area in memory to one or more areas within the computer, in accordance with the rules of editing.

A simple MOVE causes the data represented by identifier-1, or the specified literal, to be moved to identifier-2. The data is also moved to identifier-3, identifier-4, etc., if these areas are specified.

The use of this verb does not destroy the contents of the source area (identifier-1 or literal) but the receiving area (identifier-2, identifier-3, etc.) is replaced by the data of the source area.

If the CORRESPONDING option is used, selected items subordinate in hierarchy to identifier-1 are moved, with any required editing, to selected items subordinate in hierarchy to identifier-2. Items are selected on the basis of matching identifiers. Identifiers match if they and all their possible qualifiers up to, but not including identifier-1 and identifier-2 are the same. At least one of the pair of selected items must be an elementary item. In determining which identifiers are CORRESPONDING, any identifiers subordinate to identifier-1 or identifier-2 which have REDEFINEs clauses are ignored, as well as any identifiers which are subordinate to the REDEFINEd identifiers. This restriction does not preclude identifier-1 or identifier-2 themselves from having REDEFINEs clauses or from being subordinate to identifiers with REDEFINEs clauses. Identifier-1 or identifier-2 cannot have a level number of 66, 77, or 88.

Diagnostics are given for all syntactical errors in the CORRESPONDING statement. An error diagnostic is also given if identifier-1 or identifier-2 is subscripted.

It is illegal to MOVE a group item whose format is such that editing would be required on the elementary items in separate operations. If such a MOVE is desired, each elementary item must be MOVEd and edited, individually, or the CORRESPONDING option should be used.

When moving group items without the CORRESPONDING option, the move is from left to right. If the item PICTURE or USAGE is not identical, a diagnostic message is given. Truncation of low-order positions from the source item occurs if the receiving area is smaller. Space fill of low-order positions of the receiving area occurs if the source item is smaller. If the source group is computational and the receiving group is not (or vice versa), a diagnostic message is given and the above rule on size governs the results of the move.

When both the receiving and source areas are elementary items, editing, as specified in the receiving area, is automatically performed for each MOVE command. The rules governing this are:

- For Numeric Elementary Items

(a) If the source area is larger than the receiving area, truncation occurs. (Truncation is the dropping off of excessive digits.) If the receiving area is larger than the source area, the unfilled positions are zero filled. Data from the source area is aligned with respect to the implied or actual decimal point in the receiving area, with truncation or zero fill occurring to either side of the decimal point as illustrated below.

SOURCE AREA		RECEIVING AREA	
Picture	Value	Picture	Value after Moving
9V9	12	99V99	0120
9V999	8765	V99	76

(b) Data is converted from the USAGE shown in the source area to the USAGE of the receiving area (e.g., COMPUTATIONAL to DISPLAY, etc.).

(c) Insertion of a currency sign, a decimal point, commas, etc., with proper alignment, is accomplished in accordance with the PICTURE of the receiving area. If these latter characters are in a source area, the field(s) will be nonnumeric and, thus, MOVEment must conform to the nonnumeric rules.

(d) If no decimal point has been specified, either assumed or actual, data is right justified in the receiving area.

(e) A numeric edited, alphanumeric edited, or alphabetic data item must not be MOVEd to a numeric or numeric edited data item.

(f) A numeric or numeric edited data item must not be MOVEd to an alphabetic item.

(g) A numeric item whose assumed deeimal point is not to the extreme right must not be MOVEd to an alphanumeric or alphanumeric edited data item.

- For Nonnumeric Elementary Items

(a) Data from the source area is placed in the receiving area filling from left to right unless specified otherwise (e.g., JUSTIFIED RIGHT).

(b) If the receiving area is larger than the source area, the unfilled low-order positions are replaced with blanks (spaces).

(c) If the source area is greater in length than the receiving area, the MOVE terminates when the receiving area is filled. A warning is given at compilation time indicating this situation.

Examples:

MOVE A-FIELD TO COSTED-A-FIELD.

The contents of COSTED-A-FIELD are entirely replaced with the contents of A-FIELD.

MOVE 128 TO CTR.

The numeric literal, 128, is moved to the field named CTR.

MOVE SPACE TO SIZE-CODE-SIGN IN ERRORS.

The field named SIZE-CODE-SIGN is entirely filled with spaces.

Further examples of the MOVE verb are given below.

SOURCE AREA		RECEIVING AREA	
Picture	Data in source area	Picture	Data in receiving area
9999V99	567891	9999V99	567891
9999V99	567891	9999V9	56789
9V9	78	999V99	00780
XXX	M8N	XXXXX	M8N
99V99	6789	999.99	067.89
AAAAAA	WARREN	AAA	WAR
99V99	6789	$$$99.99	$67.89

4.7.3.2. EXAMINE

Format:

Option 1:

$$\underline{\text{EXAMINE}}\ \textit{identifier}\ \underline{\text{TALLYING}} \left\{\begin{array}{l}\underline{\text{ALL}}\\ \underline{\text{LEADING}}\\ \underline{\text{UNTIL}}\ \underline{\text{FIRST}}\end{array}\right\} \textit{literal-1}\ [\underline{\text{REPLACING}}\ \underline{\text{BY}}\ \textit{literal-2}]$$

Option 2:

$$\underline{\text{EXAMINE}}\ \textit{identifier}\ \underline{\text{REPLACING}} \left\{\begin{array}{l}\underline{\text{ALL}}\\ \underline{\text{LEADING}}\\ [\underline{\text{UNTIL}}]\ \underline{\text{FIRST}}\end{array}\right\} \textit{literal-1}\ \underline{\text{BY}}\ \textit{literal-2}$$

Description:

The EXAMINE verb replaces a given character or counts the number of times it appears in a data item. The data-name being EXAMINEd cannot be COMPUTATIONAL.

Any literal used in an EXAMINE statement must consist of only one character and it must be the same class as the identifier. Each literal may be any figurative constant except ALL. The literal should be bounded by quote marks. If the description of the identifier in the Data Division specifies a CLASS that uses less than the full character set (e.g., NUMERIC or ALPHABETIC), then each literal used in an EXAMINE statement must be one of the characters in the restricted set. Thus, if the CLASS of data-name is NUMERIC, each literal used in the statement must be a numeric character.

When an EXAMINE statement is executed, the examination begins with the first (i.e., the leftmost) character of the data item and proceeds to the right. Each character in the item represented by data-name is examined in turn. If the data item being examined is numeric and signed, examination excludes the operational sign.

The TALLYING option creates an integral count (i.e., a tally) which replaces the value of a special data location called TALLY. The count represents the number of:

- Occurrences of literal-1 when the ALL option is used.
- Occurrences of literal-1 prior to encountering a character other than literal-1 when the LEADING option is used.
- Characters not equal to literal-1 encountered before the first occurrence of literal-1 when the UNTIL FIRST option is used.

When either of the REPLACING options is used (i.e., with or without TALLYING) the replacement rules are as follows:

- When the ALL option is used, then literal-2 is substituted for each occurrence of literal-1.
- When the LEADING option is used, the substitution of literal-2 terminates as soon as a character other than literal-1 or the right-hand boundary of the data item is encountered.
- When the UNTIL FIRST option is used, the substitution of literal-2 terminates as soon as the first literal-1 or the right-hand boundary of the data item is encountered.
- In Option 2, when the FIRST option is used, the first occurrence of literal-1 is replaced by literal-2.

The length of the special register, TALLY, is five digits, and it may be used anywhere in the program (i.e., ADD 3, TALLY). All literals must be single character literals.

Example:

EXAMINE DOG TALLYING ALL "L" REPLACING BY "S"

DOG		Resulting Value of TALLY
Before	**After**	
L3364	S3364	1
L3L4L6	S3S4S6	3
TYPELL	TYPESS	2

4.7.4. Sequence Control Verbs

Normally, each statement in the Procedure Division is executed consecutively in order of their appearance. This is also true of the execution of each paragraph and section. However, it is often necessary to alter this normal sequence of operations and to jump to a different point in the program to execute a number of lines of coding before the next statement in sequence can logically be operated upon. Two verbs, GO TO and PERFORM, are used to fulfill this function. In addition, two supplementary control verbs are also provided: STOP and ALTER. The STOP verb controls the termination of the program, whereas ALTER permits the GO TO statement to be modified to permit control to be transferred to different points in the program.

4.7.4.1. **GO TO**

Format:

Option 1:

GO TO [*procedure-name*]

Option 2:

GO TO *procedure-name-1* [, *procedure-name-2*] . . . , *procedure-name-n*

DEPENDING ON *identifier*

Description:

This verb permits a departure from the normal sequence of procedures by specifying a transfer of control to another point in the program. If the GO TO statement is to be ALTERed when using Option 1, the paragraph in which the GO TO statement is contained must consist solely of the GO TO statement. The paragraph-name assigned to the GO TO statement is referred to by the ALTER verb in order to modify the sequence of the program.

In Option 2, the identifier must refer to a positive integral value. The branch will be to the 1st, 2nd, . . . , nth procedure-name, as the value of identifier is 1, 2, . . . , n. If the value of identifier 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.

Example:

```
CALCULATE.
    GO TO CK-FOR-TIN-WKS.
```

The example shown is an unconditional transfer of control to CK-FOR-TIN-WKS. Any GO TO statement which is a paragraph unto itself and is in the format of Option 1, however, may be ALTERed.

```
CHANGE-CALC.
     ALTER CALCULATE TO PROCEED TO CALCULATE-TONS.
     GO TO CALCULATE-TONS.
```

The above instructions change CALCULATE (the paragraph-name of the GO TO sentence in the first example) to GO TO CALCULATE-TONS, and then, unconditionally, transfer control to the routine named CALCULATE-TONS. Thus, the next time that CALCULATE is executed, control passes to CALCULATE-TONS.

The above operation could have been accomplished in another manner. At CALCULATE, we could have had this sentence:

```
GO TO CK-FOR-TIN-WKS, CALCULATE-TONS DEPENDING
ON WKS-HIGHER-THAN-TIN.
```

WKS-HIGHER-THAN-TIN would be a Working-Storage area defined with a value of 1 at compilation time. Each time this instruction was executed and WKS-HIGHER-THAN-TIN contained a 1, control would pass to CK-FOR-TIN-WKS. If control should pass to CALCULATE-TONS, a MOVE 2 TO WKS-HIGHER-THAN-TIN instruction would effect such a transfer whenever CALCULATE was executed. A subsequent MOVE 1 TO WKS-HIGHER-THAN-TIN would cause control once again to pass to CK-FOR-TIN-WKS. If anything other than 1 or 2 was moved to WKS-HIGHER-THAN-TIN, then control would "fall through" to the statement following CALCULATE.

4.7.4.2. ALTER

Format:

```
ALTER procedure-name-1 TO [PROCEED TO] procedure-name-2

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

Description:

This verb modifies the effect of a GO TO statement thereby changing the predetermined sequence of instructions.

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

Examples:

```
EX-HDR.
    GO TO SET-MON.
SET-MON.
    .
    .
    .
EX-HDR-A.
    GO TO EX-HDR-REST.
EX-HDR-REST.
    ALTER EX-HDR-A TO PROCEED TO SET-UP-EX EX-HDR
    TO PROCEED TO EX-HDR-MV.
```

The two GO TO paragraphs might have been switches originally set to fall through to the next paragraph in sequence. After being ALTERed, EX-HDR-A will show GO TO SET-UP-EX, causing control to pass to the procedure-name SET-UP-EX instead of EX-HDR-REST. Upon completion of a specified routine, EX-HDR-A may then be ALTERed back to its original status, or to some other operand by subsequent ALTER verbs. Note that EX-HDR and EX-HDR-A are one sentence paragraphs containing a GO TO statement.

Control is transferred to the next statement in sequence, following the execution of the ALTER statement.

4.7.4.3. PERFORM

Format:

Option 1:

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

Option 2:

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

$\left\{\begin{array}{l}\textit{identifier}\\ \textit{integer}\end{array}\right\}$ TIMES

Option 3:

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

UNTIL *condition*

Option 4:

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

VARYING *identifier-1* FROM $\left\{\begin{array}{l}\textit{literal-1}\\ \textit{identifier-2}\end{array}\right\}$

BY $\left\{\begin{array}{l}\textit{literal-2}\\ \textit{identifier-3}\end{array}\right\}$ UNTIL *condition*

Description:

This verb allows a temporary departure from the normal sequence of procedures in order to execute one statement or a sequence of statements a specified number of times or until a condition is satisfied and provides automatic return to normal sequence.

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

- If procedure-name-1 is a paragraph-name, and procedure-name-2 is not specified, then the return mechanism is inserted after the last statement of the procedure-name-1 paragraph.
- If procedure-name-1 is a section-name, and procedure-name-2 is not specified, then the return mechanism is inserted after the last statement of the last paragraph of the procedure-name-1 section.
- If procedure-name-2 is specified and is a paragraph-name, then the return mechanism is inserted after the last statement of the procedure-name-2 paragraph.
- If procedure-name-2 is specified and is a section-name, then the return mechanism is inserted after the last statement of the last paragraph of the procedure-name-2 section.

When procedure-name-2 is specified, the required relationship between procedure-name-1 and procedure-name-2 is that of logical sequence; i.e., execution sequence must proceed from procedure-name-1 to the last statement of the procedure-name-2 paragraph or section. GO TO statements and other PERFORM statements are permitted between procedure-name-1 and the last statement of procedure-name-2, provided the sequence ultimately returns to the final statement of procedure-name-2. If the logic of a procedure requires a conditional exit prior to the final sentence, the EXIT verb is used in order to comply with the foregoing requirements. In this case, procedure-name-2 must be the name of a paragraph consisting solely of the verb EXIT; all paths must lead to this point.

When ELSE NEXT SENTENCE appears in the last sentence (to be executed) control returns to the statement following the PERFORM statement.

A procedure referenced by one PERFORM statement can be referenced by other PERFORM statements.

PERFORM may reference a NOTE; no action is taken and the automatic return to the proper line is generated.

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 around the return mechanism to the following statement as if no PERFORM has existed.

A simplified illustration of how the bypass works is presented below. For discussion purposes, the bypass and the return mechanism are one and the same. Procedure-names have been placed on the same line as procedural-sentences for brevity.

	Procedure-Name	Procedural-Sentence	
1.	SUBROUTINE-1	READ . . .	
2.		MOVE . . .	
3.		MULTIPLY . . .	
		GO TO SUBROUTINE-2	*BYPASS-1*
4.	SUBROUTINE-2	SUBTRACT	
		GO TO SUBROUTINE-3	*BYPASS-2*
5.	SUBROUTINE-3	WRITE . . .	
.		.	
.		.	
.		.	
22.	MAIN-ROUTE	PERFORM SUBROUTINE-1.	
23.		ADD . . .	
.		.	
.		.	
.		.	
56.	GO-AGAIN	PERFORM SUBROUTINE-1 THRU SUBROUTINE-2.	
57.		GO TO . . .	
58.	ABLE	MOVE . . .	
.		.	
.		.	
.		.	
74.		PERFORM SUBROUTINE-1 THRU SUBROUTINE-2.	

At object time, BYPASS-1 is initially set to the next procedural statement (i.e., GO TO SUBROUTINE-2). The statements in italics are implied GO TO statements. Upon execution of MAIN-ROUTE, the following steps occur:

- The line number of the statement following the PERFORM (i.e., line 23) is placed in BYPASS-1.
- Control is transferred to SUBROUTINE-1.
- At the completion of SUBROUTINE-1 (lines 1 through 3), BYPASS-1 is reset to its initial status after control has transferred to line 23.

Similarly, upon execution of GO-AGAIN, BYPASS-1 will retain its GO TO SUBROUTINE-2 status, but BYPASS-2 will be changed, in effect, to GO TO line 57, i.e., the statement immediately following GO-AGAIN. (The programmer does not concern himself with these bypasses as they are automatically created by the compiler.) At times, it is necessary to give a sentence a procedure-name in order to control the number of instructions to be executed by a PERFORM sentence.

One important point should be noted in the use of the PERFORM verb: the last statement referred to in procedure-name-1, procedure-name-2, etc., must not contain a GO TO statement or unconditional (GO TO) transfers of control within conditional (IF) statements. For example, if line 3 of the illustration had GO TO ZILCH instead of MULTIPLY, then any attempt to PERFORM SUBROUTINE-1 would not be successfully concluded since control would always be transferred at line 3 to ZILCH; the bypass would never be entered and a return to the statement following the PERFORM (i.e., line 23) could never be accomplished.

Essentially then, the programmer has the facility to use conditional (IF) statements within a range of instructions referenced by a PERFORM verb. He must be governed by the fact that the bypass is only reset (i.e., returned to its original status) after the last sentence in the range has been executed. Therefore, if control is passed from the routine and never returned back into the range, the bypass will not be reset, but rather, will be set to transfer to the sentence following the last PERFORM that affected the routine. By closing the routine containing such a condition, the proper setting of the bypass can be assured.

Option 1:

In the previous example, MAIN-ROUTE and GO-AGAIN are examples of Option 1, the simple PERFORM. Briefly stated, a procedure referenced by the simple PERFORM statement is executed once and then control passes to the next statement after the PERFORM.

Example:

```
WR–EX.
    WRITE EXCEPTIONS.
MV–1–EX–PG.
    .
    .
    PERFORM WR–EX.
```

The PERFORM WR–EX causes paragraph WR–EX to be executed once and control passes to the line after PERFORM WR–EX. All necessary program steps to accomplish this are generated by the compiler.

Option 2:

The TIMES option provides a means of repeating a procedure a specified number of times. The number of times, whether stated as integer or as identifier, must be a positive integer and can be zero. If zero, no execution will occur. 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.

Example:

```
PERFORM INCREASE-WORKS 4 TIMES.
PERFORM INCREASE-WORKS CALCULATE TIMES.
```

Both of the above illustrations would give the same result assuming that CALCULATE contained the value of four. The paragraph INCREASE-WORKS is executed four times and then control passes to the statement following the PERFORM.

Option 3:

The UNTIL option is essentially the same as the TIMES option, except that the PERFORM evaluates a specified conditional expression instead of counting and testing the count against a specified integer.

Condition may be any conditional statement; that is, the condition may involve relations and tests. Condition is evaluated before execution of procedure-name-1. If the condition is false, control passes to procedure-name-1, the procedure is executed once, and control returns for another evaluation of condition. This is repeated until the condition is satisfied (i.e., true), at which time control passes to the sentence following the PERFORM. Should the conditional expression be true when the PERFORM is entered, no transfer of control to procedure-name-1 takes place and control falls through the PERFORM instruction to the next sequential sentence.

Example:

```
PERFORM READ-MASTER-ROUTINE UNTIL WORKS-CODE
OF MASTER IS EQUAL TO WORKS-CODE IN DETAIL.
```

READ-MASTER-ROUTINE will be performed until an equality is found between the two designated WORKS, at which time control will pass to the sentence following the PERFORM.

Option 4:

The VARYING option is used to PERFORM a procedure repetitively, increasing or decreasing the value of identifier-1 for each repetition, until a specific conditional expression (condition) is satisfied. Option 4 is arithmetic in nature and the arithmetic rules apply.

The PERFORM mechanism first sets the value of identifier-1 equal to its starting value (the FROM value), then evaluates the conditional expression (the UNTIL condition) for truth or falsity. If the expression is true at this point, no execution of the procedure takes place. Instead, control is transferred to the next statement after the PERFORM. If the condition is false, procedure-name-1 through procedure-name-2 are executed once, after which the PERFORM alters the value of identifier-1 by specified increment or decrement (the BY value) and again evaluates the condition for truth or falsity.

This cycle continues until the conditional expression becomes true, at which point control passes to the sentence following the PERFORM. The value of the BY and FROM clauses must be numeric, but not necessarily integral (e.g., may be 25 or V25 or .25). The initial value (FROM) must be positive or zero but the BY value may be positive, negative, or zero. After the condition is found to be true, identifier-1 will be one increment (or decrement) greater (or less) than its last used value unless the starting value (FROM value) is zero.

An extension of this option is discussed in ''Fundamentals of COBOL – Table Handling''.

''Nesting'' (i.e., PERFORM statements appearing within a sequence of statements referenced by another PERFORM) is permitted provided there is no improper overlapping. That is, the procedure associated with the included PERFORM must be either totally included in, or entirely excluded from, the procedures related to the outer PERFORM. Total inclusion requires that the entrance paragraphs be different for the external and internal PERFORM, and that the exit paragraphs be different paragraphs. The following illustrations depict correct and incorrect nesting.

Correct Nesting

```
┌ 26.
│ 27.  PERFORM 71 THRU 78.
│ 28.
│┌29.
││30.
│└31.
└ 32.
   .
  71.
   .
   .
  74.  PERFORM 29 THRU 31.
   .
  78.
   .
  87.  PERFORM 26 THRU 32.
   .
   .
   .
 102.  PERFORM 119 THRU 121.
   .
   .
   .
┌ 119.
│┌120.
└│121.
 │122.
 └123.
   .
   .
   .
 152.  PERFORM 120 THRU 123.
```

Incorrect Nesting

```
┌ 26.
│ 27.  PERFORM 29 THRU 31.
│ 28.
│┌29.
└│30.
 └31.
  32.
   .
   .
   .
  64.  PERFORM 26 THRU 30.
   .
   .
   .
┌ 70.
│ 71.  PERFORM 72 THRU 75.
│┌72.
││73.
││74.
│└75.
└
   .
   .
   .
  92.  PERFORM 70 THRU 75.
   .
   .
   .
 110.  PERFORM 72 THRU 75.
```

In the case of the incorrect nesting, the error lies in PERFORMing 29 thru 31 (Line 27) while in process of PERFORMing 26 thru 30 (Line 64). The inner nested paragraphs 29 thru 31 will not be executed to completion, since the Line 64 PERFORM has set an exit line following Line 30 (paragraph 30). Consequently, control will pass at this point (end of 30) back to the line following 64.

4.7.4.4. STOP

Format:

$$\underline{\text{STOP}}\left\{\begin{array}{l}\textit{literal}\\ \underline{\text{RUN}}\end{array}\right\}$$

Description:

The STOP verb terminates the object program either permanently or temporarily. The format must specify a literal, or the key word RUN must be used with STOP. (If a literal is employed it is displayed by the object program at the time STOP occurs either on the console printer or a substitute medium, such as console lights.) If the operator should elect, continuation of the object program begins with the execution of the next statement in sequence.

STOP RUN automatically activates the standard ending routine of the Executive Routine. Therefore, it should be used only as the final executable statement of the program.

Some examples of the use of STOP are:

```
STOP 3.
STOP 127.
STOP "INPUT TAPE SHOULD BE DESCRIPTIONS".
STOP RUN.
```

Whenever a numeric literal is used, as in the first two examples, it is customary to specify a different number for each STOP. These numbers are then catalogued with their respective definitions, for use with the object program. The literal used may be numeric or nonnumeric or any figurative constant except ALL.

4.7.5. Compiler-Directing Verbs

Certain verbs direct the compiler to perform some specific action and do not directly produce any object coding. These are called compiler-directing verbs. Two of the verbs, ENTER and EXIT, only affect the object program indirectly, whereas the verb NOTE has absolutely no effect on the object program.

4.7.5.1. EXIT

Format:

EXIT.

Description:

The EXIT verb provides an end point for a procedure being executed by a PERFORM verb. EXIT must be the only word within a paragraph.

The EXIT verb is necessary only if there is more than one common ending point to the subroutine; in this case, each of these points should come together at the EXIT in order to provide one ending point path.

EXIT should contain a transfer of control to the first sentence of the paragraph following the PERFORM paragraph. If the EXIT paragraph is referenced by some procedure other than a PERFORM, control passes through the EXIT point to the first sentence of the paragraph following the EXIT paragraph.

For example:

```
REORDER-STOCK.
            .
            .
            .
   WRITE REPLENISH-STOCK-ITEM BEFORE ADVANCING 3 LINES.
   ADD 1 TO LINE-NO.
   IF LINE-NO IS GREATER THAN 17 PERFORM HEADER-PRINT.
GO TO PARAEXIT.
HEADER-PRINT.
            .
            .
            .
            .
            .
PARAEXIT.
   EXIT.
```

PARAEXIT provides the needed common ending point for two alternate paths in the program. If LINE-NO is not greater than 17, the end of the page has not been reached, and the page heading is not required. Control falls through the IF statement to the unconditional GO TO statement. Control is transferred to PARAEXIT by the GO TO and HEADER-PRINT is not performed. If LINE-NO is greater than 17, a new page is needed, and HEADER-PRINT is performed. Control then transfers back to the statement following the PERFORM, i.e., GO TO PARAEXIT. Control now transfers to PARAEXIT and the program continues.

4.7.5.2. NOTE

Format:

NOTE *character-string*.

Description:

This verb permits the programmer to insert comments in his source program for reference purposes. These comments are printed out but have no effect on the object program.

If a NOTE sentence is the first sentence of a paragraph (i.e., immediately following a procedure-name), the entire paragraph is considered as commentary. If a NOTE sentence appears as other than the first sentence of a paragraph, the commentary ends with the appearance of a period.

Any characters from the COBOL character set may be used excluding the period which is used to terminate the comment.

4.7.5.3. ENTER

Format:

ENTER *language-name* [*routine-name*].

Description:

This verb allows the inclusion of object code produced by some assembler or compiler other than COBOL into the object program. The language-name may refer to any other language allowable on that particular computer. The language statements are executed in the object program as if they had been compiled immediately following the ENTER statement.

If the statements of the entered language cannot be written in-line (incorporated among the normal COBOL statements), a routine-name is specified to identify and access a coded portion of the entered language and to execute this coding at the desired point in the procedural sequence. The routine-name is a COBOL data-name which may only be referenced in an ENTER statement.

If the entered-language statements can be written in-line, the routine-name option is not used. The sentence:

```
ENTER COBOL
```

must follow the last statement in the entered language code segment to indicate to the compiler where a return to COBOL source language occurs.

The implementor must specify all details as to how the other languages are to be written.

4.8. SAMPLE PROBLEM

The data having been defined in a previous section, the actions to be taken at object time are now specified. These actions begin with the opening of the several files (i.e., initializing the files so as to make them capable of releasing or accepting data). Data is then read in a specific sequence, with various manipulative, logical, and arithmetic operations performed. This done, the files are closed and the run terminated.

4.8.1. Flow Chart

A flow chart solution is shown in Figure 4–1.

An English prose interpretation of the flow chart is now presented, with the COBOL version of the solution procedures shown section-by-section. The numbers in parentheses refer to the block keying numbers on the flow chart (see 4.8.2).

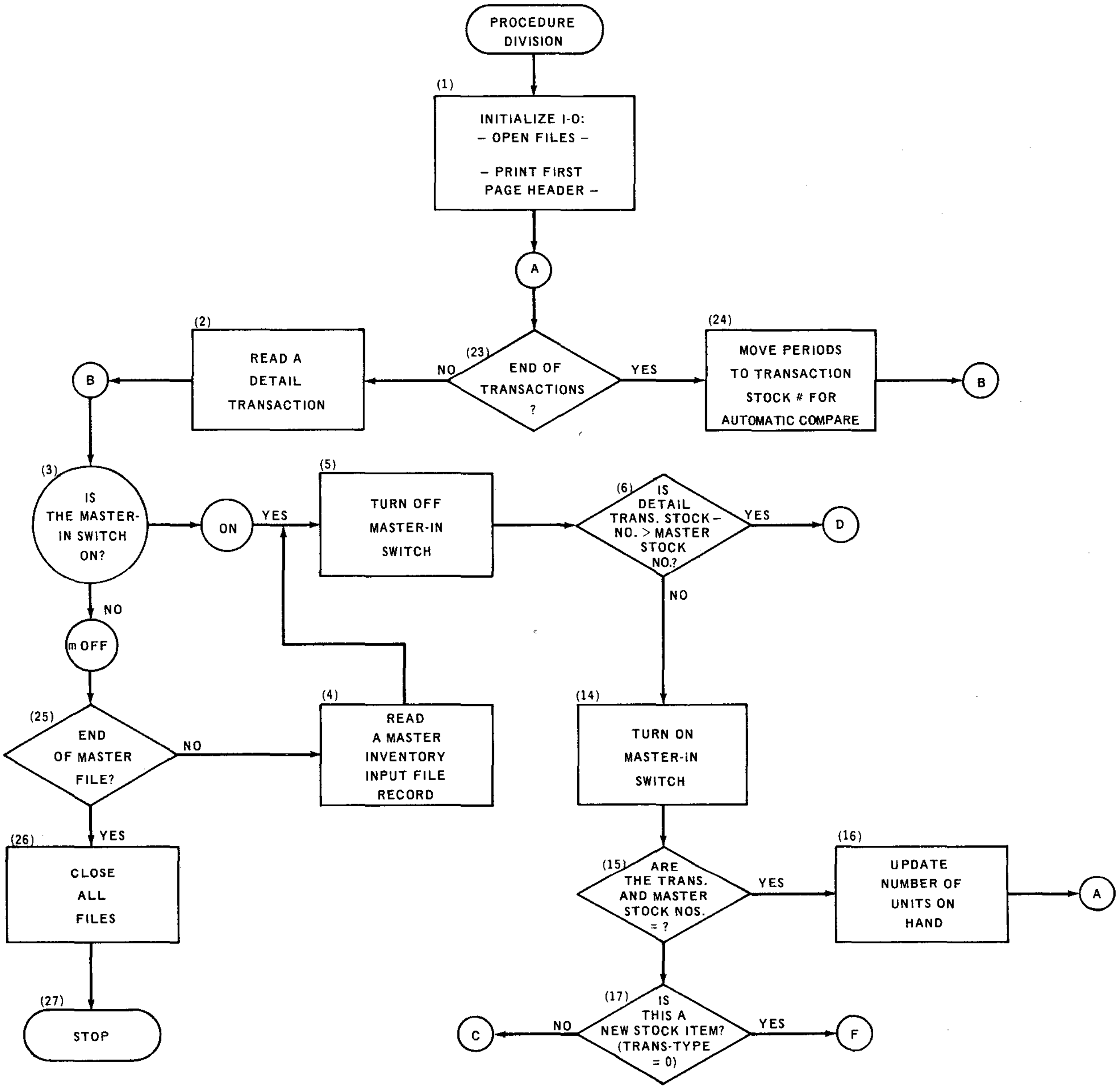

PROCEDURE DIVISION
(1)
INITIALIZE I-O:
– OPEN FILES –
– PRINT FIRST PAGE HEADER –
A
(2)
READ A DETAIL TRANSACTION
(23)
END OF TRANSACTIONS ?
NO
YES
(24)
MOVE PERIODS TO TRANSACTION STOCK # FOR AUTOMATIC COMPARE
B
B
(3)
IS THE MASTER-IN SWITCH ON?
ON
YES
(5)
TURN OFF MASTER-IN SWITCH
(6)
IS DETAIL TRANS. STOCK-NO. > MASTER STOCK NO.?
YES
D
NO
m OFF
(4)
READ A MASTER INVENTORY INPUT FILE RECORD
(25)
END OF MASTER FILE?
NO
YES
(14)
TURN ON MASTER-IN SWITCH
(26)
CLOSE ALL FILES
(15)
ARE THE TRANS. AND MASTER STOCK NOS. = ?
YES
(16)
UPDATE NUMBER OF UNITS ON HAND
A
(27)
STOP
(17)
IS THIS A NEW STOCK ITEM? (TRANS-TYPE = 0)
NO
C
YES
F

Figure 4–1. Sample Inventory Problem Flow Chart Solution (Sheet 1 of 2)

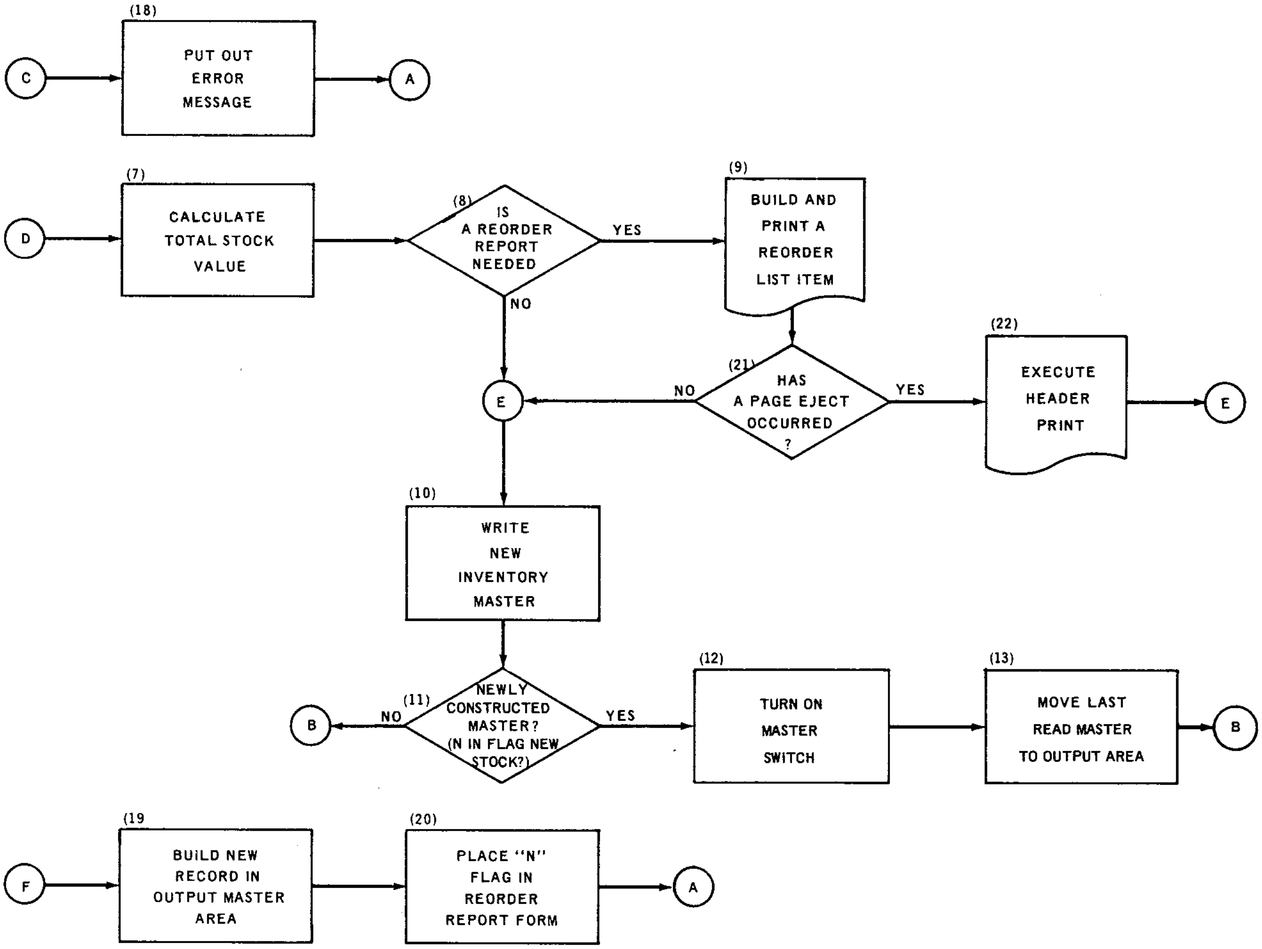

(18)
C
PUT OUT ERROR MESSAGE
A
(7)
D
CALCULATE TOTAL STOCK VALUE
(8)
IS A REORDER REPORT NEEDED
YES
NO
(9)
BUILD AND PRINT A REORDER LIST ITEM
(21)
HAS A PAGE EJECT OCCURRED ?
NO
YES
E
(22)
EXECUTE HEADER PRINT
E
(10)
WRITE NEW INVENTORY MASTER
(11)
NEWLY CONSTRUCTED MASTER? (N IN FLAG NEW STOCK?)
NO
B
YES
(12)
TURN ON MASTER SWITCH
(13)
MOVE LAST READ MASTER TO OUTPUT AREA
B
(19
F
BUILD NEW RECORD IN OUTPUT MASTER AREA
(20)
PLACE "N" FLAG IN REORDER REPORT FORM
A

Figure 4–1. Sample Inventory Problem Flow Chart Solution (Sheet 2 of 2)

4.8.2. Interpretation of Flow Chart

To initiate the run, all files are opened and the first page header is printed (1).

```
010200 PROCEDURE DIVISION.
010300 INITIALIZE.
010400     OPEN INPUT OLD-MASTER-INVENTORY, DETAIL-TRANSACTION-CARDS.
010500     OPEN OUTPUT NEW-MASTER-INVENTORY, REORDER-LIST.
010700     PERFORM HEADER-PRINT.
010800 GET-NEXT-TRANSACTION.
010900     READ DETAIL-TRANSACTION-CARDS AT END GO TO SET-UP-END-RUN.
020000 READ-INVENTORY-RECORD.
020100     IF SWITCH IS EQUAL TO 1 GO TO RESET-SWITCH.
020200     READ OLD-MASTER-INVENTORY INTO MASTER-RECORD OF
020300     NEW-MASTER-INVENTORY AT END GO TO CLOSE-FILES.
020800 RESET-SWITCH.
020900     MOVE ZERO TO SWITCH.
```

Now the relationship of the current input master inventory record and the current detail transaction is determined. When a detail transaction is read (2) (assuming there are any remaining to be read) (20) the Master-In switch is tested (3).

The Master-In switch is ON, whenever the processing of the last read input master has not yet been terminated and no new master is to be read. If OFF, a new input master must be read (4) because the processing of the last input master read has been terminated prior to the reading of the detail transaction. In either case the Master-In switch is turned off (5) immediately after testing. It is turned on at other points in the program when it develops that the master just read is not to be immediately terminated.

```
300300 HEADER-PRINT.
300400     ADD 1 TO PAGE-NO.
300500     WRITE REPLENISH-STOCK-ITEM FROM 1-REPLENISH-REPORT-HEADING
300510     AFTER ADVANCING NEW-PAGE LINES.
300600     WRITE REPLENISH-STOCK-ITEM FROM 2-REPLENISH-REPORT-HEADING.
300610     WRITE REPLENISH-STOCK-ITEM FROM 3-REPLENISH-REPORT-HEADING.
300620     MOVE SPACES TO REPLENISH-STOCK-ITEM.
300630     WRITE REPLENISH-STOCK-ITEM.
300700     MOVE ZEROS TO LINE-NO.
```

If the stock number in the newly read detail transaction is greater than the input master stock number (6) all detail transactions for this master (if any) have been processed. Complete the processing of the master (D) as follows: compute the total stock value (7) by multiplying the number of units on hand times the price per unit. Check whether the number of units on hand has fallen below the reorder level (8), and if so, write a list entry on the stock reorder report (9). Write the updated master inventory record from the output area (10). If the output master was constructed from a new stock item detail transaction (i.e., if new stock flag contains ''N'') (11), set the Master-In switch ON (12) and move the input area copy of the last input master inventory record to the output area (13). Branch to read the next input master (B).

```
030000 CHECK-STOCK-NUMBER.
030100     IF STOCK-CONTROL-NUMBER GREATER THAN SEQ-STOCK-NUMBER
030200     OF NEW-MASTER-INVENTORY GO TO FINISH-MASTER.
```

```
060100 FINISH-MASTER.
060200     MULTIPLY ON-HAND-UNITS OF NEW-MASTER-INVENTORY BY COST-PER-UN
060300-    IT OF NEW-MASTER-INVENTORY GIVING TOTAL-WHOLESALE-VALUE OF
060400     NEW-MASTER-INVENTORY ON SIZE ERROR DISPLAY 'OVERFLOW ON TOTAL
060410-    ' WHOLESALE VALUE'.
060420     DISPLAY SEQ-STOCK-NUMBER OF MASTER-RECORD IN
060430     NEW-MASTER-INVENTORY.
060500     IF FLAG-NEW-STOCK IS EQUAL TO 'N' MOVE 1 TO SWITCH.
060600     IF MIN-STOCK-UNIT-QUANTITY OF NEW-MASTER-INVENTORY IS GREATER
060700     THAN ON-HAND-UNITS IN NEW-MASTER-INVENTORY OR FLAG-NEW-STOCK
060800     IS EQUAL TO 'N' PERFORM REORDER-STOCK.
070100     PERFORM WRITE-NEW-INVENTORY.
070200     MOVE MASTER-RECORD OF OLD-MASTER-INVENTORY TO MASTER-RECORD
070210     OF NEW-MASTER-INVENTORY.
070300     GO TO READ-INVENTORY-RECORD.
070400 WRITE-NEW-INVENTORY.
070500     WRITE MASTER-RECORD OF NEW-MASTER-INVENTORY.
```

Assuming the stock number on the detail transaction was not greater, turn the Master-In switch ON (14) and test if the stock numbers are equal (15). If so, update the quantity on hand by the quantity of the shipment or receipt (16). Then go to read the next detail transaction (A).

```
030300     MOVE 1 TO SWITCH.
030400     IF STOCK-CONTROL-NUMBER IS EQUAL SEQ-STOCK-NUMBER
030500     OF NEW-MASTER-INVENTORY GO TO UPDATE.
```

```
040400 UPDATE.
040600     IF TYPE-TRANSACTION IS EQUAL TO 1 OR 2 GO TO SHIPMENT.
040700     IF TYPE-TRANSACTION IS EQUAL TO 3 OR 4 GO TO RECEIPT.
040800     GO TO ERROR-MESSAGE.
050300 SHIPMENT.
050400     SUBTRACT QUANTITY FROM ON-HAND-UNITS
050500     IN NEW-MASTER-INVENTORY.
050600     GO TO GET-NEXT-TRANSACTION.
050700 RECEIPT.
050800     ADD QUANTITY TO ON-HAND-UNITS IN
050900     NEW-MASTER-INVENTORY.
060000     GO TO GET-NEXT-TRANSACTION.
```

If the detail stock number is less than the master stock number, test whether the transaction is type 0 (17) (new stock item). If not, display an error message describing the transaction (18). If so, use the detail transaction to build a new master inventory record (19) and set the new-stock-item flag (20) in the output area. In either case, go to read the next transaction (A).

```
030900     IF TYPE-TRANSACTION IS EQUAL TO ZERO GO TO NEW-STOCK-ITEM.
040000 ERROR-MESSAGE.
040100     DISPLAY STOCK-CONTROL-NUMBER ' NOT IN FILE, TRANSACTION
040200-    ' TYPE IS  ' TYPE-TRANSACTION.
040300     GO TO GET-NEXT-TRANSACTION.



100100 NEW-STOCK-ITEM.
100500     MOVE SPACES TO MASTER-RECORD OF NEW-MASTER-INVENTORY.
100600     MOVE STOCK-CONTROL-NUMBER TO SEQ-STOCK-NUMBER IN
100700     NEW-MASTER-INVENTORY.
100800     MOVE NO-TRANSACTOR TO NUMBER-MANUFACTURER IN
100900     NEW-MASTER-INVENTORY.
200200     MOVE ORDER-NUMBER TO MFR-CATALOG-NUMBER IN
200300     NEW-MASTER-INVENTORY.
200400     MOVE DTL-DESCRIPTION TO DESCRIPTION IN NEW-MASTER-INVENTORY.
200600     MOVE ZEROS TO ON-HAND-UNITS OF NEW-MASTER-INVENTORY,
200700     TOTAL-WHOLESALE-VALUE IN NEW-MASTER-INVENTORY.
200800     MOVE UNIT-COST OF TRANSACTIONS TO COST-PER-UNIT IN
200900     NEW-MASTER-INVENTORY.
300000     MOVE QUANTITY TO MIN-STOCK-UNIT-QUANTITY IN
300010     NEW-MASTER-INVENTORY.
300100     MOVE 'N' TO FLAG-NEW-STOCK.
300200     GO TO GET-NEXT-TRANSACTION.
```

A list, called the stock-reorder list, is printed to reflect newly added stock items, and stock items on which the stock on hand falls below the minimum level. Information pertinent to the current status of the stock is shown, including a flag to indicate a newly added stock item and a flag to indicate when the stock level has fallen below 60 percent of minimum. Following each line printed, a test is made to determine whether a page eject has occurred (21). If so, the page counter is updated and the report header printed.

```
070700 REORDER-STOCK.
080000     MULTIPLY ON-HAND-UNITS OF NEW-MASTER-INVENTORY BY 100
080050     GIVING DIVIDEND ON SIZE ERROR DISPLAY
080060     'OVERFLOW ON MULTIPLICATION'.
080070     DISPLAY SEQ-STOCK-NUMBER OF MASTER-RECORD IN
080080     NEW-MASTER-INVENTORY.
080100     DIVIDE MIN-STOCK-UNIT-QUANTITY OF NEW-MASTER-INVENTORY INTO
080200     DIVIDEND GIVING PERCENTAGE ROUNDED.
080300     IF PERCENTAGE IS LESS THAN 60 MOVE 'EEE' TO
080400     EMERGENCY-REORDER-FLAG.
080500     MOVE SEQ-STOCK-NUMBER OF NEW-MASTER-INVENTORY TO
080600     STK-NUMBER-PRINT.
080700     MOVE NUMBER-MANUFACTURER OF NEW-MASTER-INVENTORY TO NO-MFR.
080900     MOVE MFR-CATALOG-NUMBER OF NEW-MASTER-INVENTORY TO
090000     MFR-ORDER-NUMBER OF REORDER-LIST.
090100     MOVE DESCRIPTION IN NEW-MASTER-INVENTORY TO ITEM-DESCRIPTION.
090200     MOVE ON-HAND-UNITS IN NEW-MASTER-INVENTORY TO UNITS-ON-HAND.
090400     MOVE MIN-STOCK-UNIT-QUANTITY OF NEW-MASTER-INVENTORY TO
090500     MIN-UNITS.
090600     MOVE COST-PER-UNIT IN NEW-MASTER-INVENTORY TO UNIT-COST
090610     OF REORDER-LIST.
090700     WRITE REPLENISH-STOCK-ITEM BEFORE ADVANCING 3 LINES.
090800     ADD 1 TO LINE-NO.
090810     MOVE SPACES TO REPLENISH-STOCK-ITEM.
090900     IF LINE-NO IS GREATER THAN 17 PERFORM HEADER-PRINT.
```

After all detail transactions have been read and processed (23), the detail transaction input area is modified so that all subsequent input master inventory records to be read will have lower stock numbers than that shown in the detail transaction field (24). Thus, all remaining input master inventory records will be automatically transferred to the output master file without updating.

```
300900 SET-UP-END-RUN.
400000     MOVE '......' TO STOCK-CONTROL-NUMBER.
400100     GO TO READ-INVENTORY-RECORD.
```

When the end of the input master inventory file is reached and the last input master record put on the output master inventory file (25), all files are closed (26) and the object run is terminated (27).

```
400200 CLOSE-FILES.
400300     CLOSE OLD-MASTER-INVENTORY, DETAIL-TRANSACTION-CARDS.
400690     CLOSE NEW-MASTER-INVENTORY, REORDER-LIST.
400700     STOP RUN.
```

5. DESCRIBING THE EQUIPMENT AND PHYSICAL ENVIRONMENT

5.1. GENERAL DESCRIPTION

In the section called the *Environment Division* of a COBOL source program a relationship is established between the physical requirements of the computing system on which the program will operate and the operations to be performed. That is, the Environment Division describes the computing system on which the object program is to run so that the succeeding divisions of the source program can be translated to an object program for that computing system.

The Environment Division is completely machine oriented and must be rewritten whenever a source program is to be translated to operate on a different computing system. It may be advisable, or necessary, to rewrite this division for different configurations of the computing system for which the source program is to be translated.

5.2. ORGANIZATION AND STRUCTURE OF THE ENVIRONMENT DIVISION

The Environment Division comprises two sections, each of which has a fixed name. They are:

CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.

The following is a general outline of the sections and paragraphs contained in the Environment Division and their order of presentation:

```
A    B
8    12
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.......
OBJECT-COMPUTER.......
SPECIAL-NAMES.......
INPUT-OUTPUT SECTION.
FILE-CONTROL. file-control entry
I-O-CONTROL. input-output-control entry
```

The discussions that follow each of the Environment Division paragraphs are general in nature. Information relating to the specifics of implementation can be found in the programmers reference manual pertaining to the particular computer.

5.3. CONFIGURATION SECTION

This section details the overall specifications for the computing systems involved in a COBOL program. It comprises three paragraphs as follows:

(a) SOURCE-COMPUTER – Defines the computer on which the COBOL-language program is to be compiled.

(b) OBJECT-COMPUTER – Defines the computer on which the compiled object program is to be run.

(c) SPECIAL-NAMES – Defines problem-oriented names for specific pieces of equipment.

5.3.1. SOURCE-COMPUTER

Format:

SOURCE-COMPUTER. *computer-name*.

Description:

This paragraph names the computer upon which the source program is to be compiled and provides a means of communicating with the executive routine.

5.3.2. OBJECT-COMPUTER

Format:

$\underline{\text{OBJECT-COMPUTER}}$. *computer-name*

$$\left[\ \underline{\text{MEMORY}}\ \text{SIZE}\ \textit{integer} \left\{ \begin{array}{l} \underline{\text{WORDS}} \\ \underline{\text{CHARACTERS}} \\ \underline{\text{MODULES}} \end{array} \right\} \right]$$

Description:

This paragraph describes the computer upon which the object program is to be run and specifies the equipment configuration present during object running time.

Computer-name provides a means for describing equipment configuration. The computer-name and its implied configuration are specified by each implementor. If the configuration implied by computer-name comprises more or less equipment than is actually needed by the object program, the descriptive clauses following computer-name permit the specification of the actual subset of the configuration. The configuration definition contains specific information concerning the MEMORY SIZE.

The implementor should define what must be done if the configuration specified by the user is less than the minimum configuration required to run the object program.

5.3.3. SPECIAL-NAMES

Format:

```
SPECIAL-NAMES.

┌                   ⎧IS mnemonic-name-1 [, ON STATUS IS condition-name-1
│ implementor-name ⎨ IS mnemonic-name-2 [, OFF STATUS IS condition-name-3
│                   ⎪ON STATUS IS condition-name-5
└                   ⎩OFF STATUS IS condition-name-7

 [, OFF STATUS IS condition-name-2]]⎫ ┐
 [, ON STATUS IS condition-name-4]] ⎪ │
 [, OFF STATUS IS condition-name-6] ⎬ │ . . .
 [, ON STATUS IS condition-name-8]  ⎭ ┘

 [, CURRENCY SIGN IS literal] [, DECIMAL-POINT IS COMMA].
```

Description:

This paragraph provides a means of relating hardware device names (implementor-name) to problem-oriented mnemonic names assigned by the user. This paragraph is usually not required if the actual hardware names are used in the program. A mnemonic-name may only be used in formats which specifically permit their use. If the implementor-name is not a switch, the associated mnemonic-name may be used in ACCEPT, DISPLAY, and WRITE statements.

If implementor-name is a switch, it must be assigned a mnemonic-name, a condition-name, or both. The status of a switch is interrogated by testing the condition-name assigned to it (see 4.2.2.4).

The literal specified in the CURRENCY IS clause is used in PICTURE clauses to represent the currency symbol. This literal must be a single character and it cannot be any of the following:

- Digits: 0 through 9.
- Alphabetics: A, B, C, D, P, R, S, V, X, Z, or the space.
- Special characters:

  ```
  *  ,  (
  +  .  )
  –  ;  ''
  ```

When the CURRENCY IS clause is not present, only the currency symbol specified by the implementor may be used in PICTURE clauses.

The DECIMAL-POINT IS COMMA clause interchanges the functions of the comma and period in PICTURE clause character-strings and in numeric literals.

An example of the use of the SPECIAL-NAMES feature is as follows:

SPECIAL-NAMES. CARD-READER IS READER.

This paragraph, specified in the Environment Division, would permit reference to the hardware device CARD-READER to be made in the program in the following manner:

ACCEPT RATE-CHANGE FROM READER.

The individual programmers reference manual must be consulted for both the permitted use of mnemonic-names, as well as the standard hardware device names for that computer.

5.4. INPUT-OUTPUT SECTION

This section deals with the definition of the external media and provides information needed to create the most efficient transmission of data between the media and the object program. This section is divided into two paragraphs as follows:

(a) FILE-CONTROL – Names and associates the files with external media.

(b) I-O-CONTROL – Defines special input/output techniques, rerun, and multiple-file tapes.

5.4.1. FILE-CONTROL

Format:

$$\underline{\text{FILE-CONTROL}}.\left\{\begin{array}{l} \underline{\text{SELECT}}\ [\underline{\text{OPTIONAL}}]\ \textit{file-name} \\ \underline{\text{ASSIGN}}\ \text{TO}\ [\textit{integer-1}]\ \textit{implementor-name-1}\ [,\ \textit{implementor-name-2}]\ \ldots \\ \left[\text{FOR}\ \underline{\text{MULTIPLE}}\ \underline{\text{REEL}}\right] \left[,\ \underline{\text{RESERVE}} \left\{\begin{array}{l}\textit{integer-2}\\ \underline{\text{NO}}\end{array}\right\} \text{ALTERNATE} \left[\left\{\begin{array}{l}\text{AREA}\\ \text{AREAS}\end{array}\right\}\right]\right]\ . \end{array}\right\}\ \ldots$$

Description:

This paragraph is used to name each file; identify the hardware medium which contains it; permit specific hardware assignments for the program; and to specify alternate input/output areas.

Each file selected by a FILE CONTROL entry must have a corresponding File Description entry in the Data Division.

The keyword OPTIONAL is required for input files that may or may not be present when the object program is run.

Integer-1 indicates the number of input-output units assigned to a file-name. If integer-1 is not specified, the compiler determines the number of units to be assigned.

All files employed in the program must be ASSIGNed to a specific input/output hardware device (card reader, tape unit, printer, etc.). The exact name for each of these units will be covered in the specific programmers reference manual. If a tape file is to use more than one reel, the FOR MULTIPLE REEL option is specified.

The RESERVE clause allows the user to modify the number of input/output areas allocated by the compiler. The option RESERVE integer ALTERNATE AREAS means that integer additional areas are to be reserved for the file in addition to the minimum area. The particular implementation specifies the minimum area and the additional areas for particular hardware. No additional areas are reserved for the file when the NO option is selected.

5.4.2. I-O-CONTROL

Format:

```
I-O-CONTROL.
┌                                                               ┐
│ RERUN [ON { file-name-1        } ]                            │
│           { implementor-name   }                              │
│                                                               │ . . .
│        ( { END OF REEL        }                 )             │
│        ( {                    } OF file-name-2  )             │
│  EVERY ( { integer-1 RECORDS  }                 )             │
│        (   integer-2 CLOCK-UNITS                )             │
│        (   condition-name                       )             │
└                                                               ┘
[; SAME [RECORD] AREA FOR file-name-3 {, file-name-4} . . . ] . . .

[; MULTIPLE FILE TAPE CONTAINS file-name-5 [POSITION integer-3]

[, file-name-6 [POSITION integer-4]] . . . ]. . . .
```

Description:

This paragraph, which is optional, permits the user to specify such things as input/output techniques and points at which rerun is to be established.

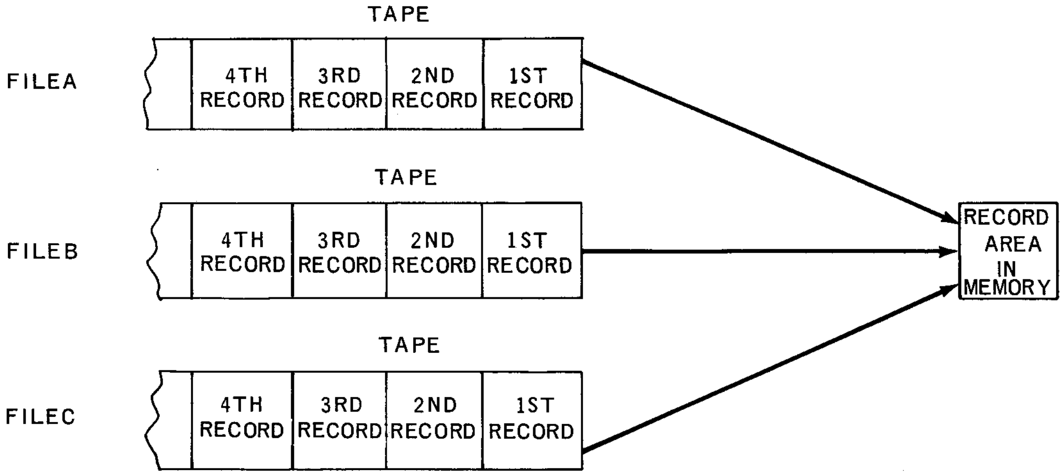
When either the integer-1 RECORDS or integer-2 CLOCK-UNITS option is selected, the implementor-name must be given in the RERUN option.

The RERUN clause specifies where the rerun information is recorded and when the memory dump occurs. Memory dumps may be recorded in either of the following ways:

(a) The memory dump is written on each reel or unit of an output file with the specific implementation specifying where, on the reel or file, the dump is to be recorded.

(b) The memory dump is written on a separate rerun tape or unit, as specified by the hardware-device name of the RERUN option.

The SAME AREA clause specifies that two or more files are to use the same memory area (including alternate areas) during processing. If the RECORD option is specified, only one record can reside in the record area at any one time regardless of the number of files that may be open (see diagram below). Since only one record can be in the record area, it is the responsibility of the programmer to determine which record of which file is in the record at any point in the program.



If the RECORD option is not specified in the SAME AREA clause, then the area being shared includes all storage assigned to the files specified in the SAME AREA clause. In this case, the programmer must make certain that no more than one file is open at any point in the program.

Since more than one SAME AREA or SAME RECORD AREA clause can be written in the I-O-CONTROL paragraph, certain restrictions are placed on the use of the file-names. These restrictions are as follows:

(1) A particular file-name cannot be specified in more than one SAME AREA clause nor in more than one SAME RECORDS AREA clause.

(2) If one or more of the file-names specified in SAME AREA clause also appears in the SAME RECORDS AREA clause, all file-names specified in the SAME AREA clause must be listed in the SAME RECORDS AREA clause. However, any dissimilar file-names specified in the SAME RECORDS AREA clause are not be be listed in the SAME AREA clause. For example, assume that FILEA, FILEG, and FILEL are to have the same area and that FILEA, FILE B, and FILEZ are to have the same record area. The SAME AREA and SAME RECORD AREA clauses must be written as follows:

```
SAME AREA FILEA, FILEG, FILEL
SAME RECORD AREA FILEA, FILEB, FILEG, FILEL, FILEZ
```

(3) If both a SAME AREA clause and a SAME RECORD AREA clause appear in an I-O-CONTROL paragraph, only one file can be open at any point in the program.

The MULTIPLE FILE option is required when more than one file shares the same physical reel of tape. Regardless of the number of files on a single reel, only those files used in the object program need be specified. The POSITION option is not required if all the files are listed in consecutive order. If any file on the tape is not listed, the position relative to the beginning of the tape must be given for each file listed.

5.5. SAMPLE PROBLEM

The Environment Division section of the program describes and centralizes those aspects of the total problem which depend upon the characteristics of the computing equipment; it links the logical concepts of data and records described in the Data Division with the physical aspects of the files on which they are stored. The various information will be shown first in English prose and then in COBOL. The header for this division is:

```
001400 ENVIRONMENT DIVISION.
```

5.5.1. Configuration Section

```
001500 CONFIGURATION SECTION.
```

■ Source-Computer

The source-computer (the computer on which the compiling operation will be performed) is specified in a format reserved for that particular computer.

```
001600 SOURCE-COMPUTER. MARK-I.
```

■ Object-Computer

The object-computer (the computer on which the compiled object program will be executed) is specified, and its equipment configuration and operational mode is described as follows.

– Name of Computer.

– Memory Size Used (20,000 words).

```
001700 OBJECT-COMPUTER. MARK-I.
001800     MEMORY SIZE 20000 WORDS.
```

■ Special-Names

This paragraph specifies mnemonic names which may be equated to standard hardware names or switches. In this case, TOP-OF-NEXT-PAGE could contain the computer manufacturer's symbol which instructs the printer to advance to a new page. The programmer has equated his name, NEW-PAGE, with the manufacturer's name.

```
001850 SPECIAL-NAMES. NEW-PAGE IS TOP-OF-NEXT-PAGE.
```

5.5.2. Input/Output Section

```
002100 INPUT-OUTPUT SECTION.
```

- File Control

  Each file is named and is described as to medium and hardware assignment. The files assigned in this program are as follows:

  – The old (input) master inventory file is assigned to a tape.

  – The new (output) master inventory is assigned to a tape. This file is an updated version of the old (input) master inventory, with an identical file description and record description; therefore, the RENAMING clause may be used, and it is not necessary that the two files be defined separately in the Data Division.

  – The detail transaction cards input file is assigned to the card reader.

  – The stock reorder output list file is assigned to the printer.

```
002200 FILE-CONTROL.
002300     SELECT OLD-MASTER-INVENTORY ASSIGN TO MAGNETIC-TAPE-UNIT.
002400     SELECT NEW-MASTER-INVENTORY ASSIGN TO MAGNETIC-TAPE-UNIT.
002600     SELECT DETAIL-TRANSACTION-CARDS ASSIGN TO CARD-READER.
002610     SELECT REORDER-LIST ASSIGN TO PRINTER.
```

- Input/Output Control

  In this instance, the I-O-CONTROL statement is used to obtain a memory dump at the end of each reel of the NEW-MASTER-INVENTORY file. This information is useful when restarting a program that has aborted during a run; the program can be restarted at the last valid rerun point and there is no need to rerun the entire program.

```
002640 I-O-CONTROL.
002650 RERUN EVERY END OF REEL OF NEW-MASTER-INVENTORY.
```

6. IDENTIFYING THE PROGRAM

6.1. GENERAL DESCRIPTION

In the section called the *Identification Division,* the information identifying the source program and the output of a compilation is provided. In this division, the user may include such information as the name of the program, date of compilation, programmer's name and so forth. Information provided in this division is listed on the printed output of the compilation but has no effect upon the object program.

6.2. ORGANIZATION AND STRUCTURE

Fixed paragraph names identify the type of information provided in this division. The name of the program, PROGRAM-ID, must be present. The other paragraphs are optional and may be specified at the user's discretion. The format of presentation is as follows:

```
 A   B
 8   12
 IDENTIFICATION DIVISION.
 PROGRAM-ID. program-name.
[AUTHOR. [comment-entry]. . . ]
[INSTALLATION. [comment-entry]. . . ]
[DATE-WRITTEN. [comment-entry]. . . ]
[DATE-COMPILED. [comment-entry]. . . ]
[SECURITY. [comment-entry]. . . ]
[REMARKS. [comment-entry]. . . ]
```

A comment-entry may be made up of any combination of characters from the allowable COBOL character set. The PROGRAM-ID must always appear as the first paragraph following IDENTIFICATION DIVISION. This paragraph must be a single word. It is used to identify the source program and may also identify the resulting object program. The contents of the paragraph named DATE-COMPILED will be replaced by the current date and a period.

6.3. SAMPLE PROBLEM

This section of the program identifies or labels the program. It may also contain any other documentational information that is desired as to authorship, location of installation, date of writing or compiling, security, and any other comments regarding the functional or peripheral aspects of the program. Information that may be included is shown below, first in English prose and then in the COBOL version:

- A Program Identification (required).
- Author.
- Installation.
- Date written.
- Date compiled.
- Security level of the program output.
- Remarks. A general functional description of the program.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. MASTER-INVENTORY-UPDATE.
000300 AUTHOR. AB. C. DEFGHI.
000350 INSTALLATION. OSHKOSH.
000400 DATE-WRITTEN. JUNE 7 1966.
000450 DATE-COMPILED. JUNE 23 1966.
000500 SECURITY. CLASS B COMPANY CONFIDENTIAL.
000600 REMARKS. MASTER INVENTORY ON TAPE IS RUN AGAINST DETAIL
000700     TRANSACTIONS FOR UPDATING TO CREATE A NEW MASTER
000800     INVENTORY FILE. NEW STOCK ITEMS MAY BE ADDED BY
000900     ZERO-TYPE DETAIL TRANSACTIONS.
001000     STOCK ITEMS ON WHICH THE QUANTITY ON HAND HAS
001100     FALLEN BELOW REORDER LEVEL WILL BE LISTED WITH
001200     PURCHASING INFORMATION ON THE PRINTER, AS WILL
001300     BE ALL NEW STOCK ITEMS.
```

7. COBOL REFERENCE FORMAT

7.1. GENERAL

The standard *COBOL Reference Format* consists essentially of a basic format medium within which the COBOL source program is constructed. This is necessary because the COBOL language must be used very precisely at each level of organization. Extreme care must be used to observe the rules stated in this manual.

In addition to observing the rules of sentence formats, a source program must be written such that its various sections and divisions appear in a particular sequence at compilation time.

The standard COBOL Reference Format prescribes the standards by which source program information must be arranged and sequenced on the COBOL coding form for the compiler to interpret the information and convert it to an object program that will perform the intended functions.

This section will define the purposes of the Reference Format and the rules governing its use in each of the four divisions of the COBOL source program. In doing this, the various rules and principles previously stated in this manual will be summarized.

The description of the COBOL Reference Format contained in this section is general. This is because certain differences may exist in any specific COBOL implementation for particular equipment. It is intended as a basic reference guide; however, when preparing to write a COBOL program for any specific equipment, consult the COBOL programmers reference manual for the particular system.

7.2. PURPOSE OF REFERENCE FORMAT

The Reference Format has three main purposes.

- It provides a convenient form for the programmer to use. The COBOL Programming Form helps both the programmer and the person who punches cards from this format to arrange the program in the proper form.
- It provides a medium by which the programmer may specify those items that the compiler needs to create the object program.
- It provides a standard form for the printed listing of the source program which could serve, with modification, as the input form to a COBOL compiler on another computer.

7.3. USING THE REFERENCE FORMAT

The rules that follow for the use of the Reference Format take precedence over any other rules stated in this manual with respect to spacing of formats.

7.3.1. The Reference Format Programming Form

The layout of the Reference Format Programming Form is shown in Figure 2–1. Each line of coding represents the information that is to be keypunched on one 80-column card, as indicated in the following paragraphs.

7.3.1.1. Sequence Number

Columns 1 through 6 contain the sequence number. Sequence numbers are optional, and are intended to aid the programmer in making corrections and changes in the source program by establishing a linear card sequence.

7.3.1.2. Continuation Indicator

Column 7 is used as a continuation indicator. Whenever a paragraph or entry requires more than one line of coding, *and* the break in the line occurs in the middle of a word or literal, the continuation is shown by a hyphen in column 7 of the line in which the broken word or literal is completed.

If no hyphen appears at the start of any given line, the last word of the preceding line is assumed to have ended. No space is required at the end of any line. If the hyphen is used, the first character of the hyphenated line is considered a continuation of the last word of the preceding line.

A word or numeric literal may be interrupted in any column (and the rest of the line space filled) if there is a hyphen in column 7 of the next succeeding line.

If a nonnumeric literal extends beyond the end of a line, there must be a hyphen in column 7 of the next line; also, the continuation of the nonnumeric literal must start with a quotation mark. Until the final quotation mark terminates the nonnumeric literal, it is assumed that each card column up to and including column 72 is part of the literal.

7.3.1.3. Text

Columns 8 through 72 are used to contain text, i.e., the information from which the data and instructions which comprise the object program are compiled. Two margins are used to align this information: Margin A and Margin B. An item aligned with Margin A has its first character in column 8; an item aligned with Margin B has its first character in column 12.

Names of divisions, names of sections, names of paragraphs, and all main entries of the Data Division (i.e., file descriptions and record descriptions) are placed at Margin A. Division and section names must appear on a single line.

Subordinate items, continuations, and procedural statements are placed at Margin B.

7.3.1.4. Identification

Columns 73 through 80 are used for card deck identification at the discretion of the programmer. The contents are shown in the source program listing, but do not affect the compilation operation.

7.3.2. The Reference Format Divisions

There are four divisions to the Reference Format: the Identification Division, the Environment Division, the Data Division, and the Procedure Division, appearing in the Reference Format in that order. The remainder of this section explains the format usage rules for these divisions. Sample formats may be seen in the sample program which has been developed throughout the manual and is shown in full in Appendix C.

7.3.2.1. Rules for the Identification Division

The Identification Division provides a means of identifying or labeling 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.                INSTALLATION.
DATE-WRITTEN.          DATE-COMPILED.
SECURITY.              REMARKS.
```

The name of the division, and the names of the paragraphs within it, start under Margin A. The first line of this division contains its name, followed by a period; i.e.,

```
IDENTIFICATION DIVISION.
```

The text of each paragraph may start on the same line as the paragraph name, or on the next line, as preferred. Any paragraph which occupies more than one line must be continued by starting at Margin B on the next line.

7.3.2.2. Rules for the Environment Division

The Environment Division specifies those elements of a COBOL program that are dependent upon physical aspects or limitations of the specific equipment.

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

```
ENVIRONMENT DIVISION.
```

The section-names CONFIGURATION SECTION and INPUT-OUTPUT SECTION must also be single entries, each on a line by itself and followed by a period.

Paragraph-names, like the division-name and section-name, must each start at Margin A and must be followed by a period. However, the clauses which comprise each paragraph may follow the paragraph immediately, on the same line. The I-O-CONTROL and the FILE-CONTROL paragraphs may each be comprised of several sentences, whereas the paragraphs of the CONFIGURATION section are each composed of one sentence only.

7.3.2.3. Rules for the Data Division

The basic unit in the Data Division is an entry. Each entry begins with a level indicator or level number; that is followed by a data-name, or the word FILLER and then by a set of descriptive clauses.

The first line of the Data Division is the division-name, followed by a period; i.e.,

```
DATA DIVISION.
```

The Data Division is separated into two distinct sections; the FILE SECTION and the WORKING-STORAGE SECTION. Each section is begun with the appropriate section-name; the section-name is on a line by itself followed by a period.

In the File Section, each file description is begun with its appropriate level indicator at Margin A, followed by the file-name at Margin B. File-descriptive clauses may then follow on the same line, and continue until a period ends the File Description entry.

All Record Description entries for label records or data records that pertain to a given file must follow the File Description for that file.

After all the File Section entries are completed, the Working-Storage Section entries are made. These entries must be preceded by the words: WORKING-STORAGE SECTION on a line by themselves. The level-number of the entry is aligned with Margin A, while the entry itself starts at Margin B. In the Working-Storage Section, all 77-level entries, and their subordinate 88-level condition-name entries, must precede any record description entries.

In any section of the Data Division, various entries may be indented for the purpose of displaying a hierarchal data structure within a given Record Description. The choice between left-justification or indentation of entries according to level number is left to the discretion of the programmer; under no circumstances does indentation affect the magnitude of a level number or the result of a compilation operation.

The 01 level number which begins a record description is placed at Margin A, after which the data-name for the record is placed at Margin B. All entires subordinate to the record description are begun with the level number at Margin B, with the data-name separated from the level number by one or more spaces. If a single entry requires more than one line, the left margin limit for each line within the entry is the same; i.e., the position under the first character of the data-name.

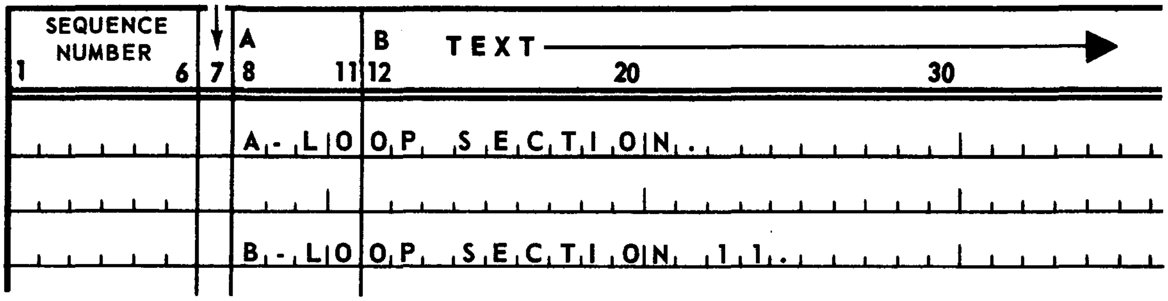
When level numbers are indented each new level number is placed four spaces to the right of the starting position of the previous level number.

7.3.2.4. Rules for the Procedure Division

The first line of the division consists of its name, followed by a period, starting at Margin A, as follows:

```
PROCEDURE DIVISION.
```

A section is designated as shown in the following examples:



```
A-LOOP SECTION.

B-LOOP SECTION 11.
```

The section-name is followed by a space, the word SECTION, and a period. However, when the use of segmentation requires that a priority-number be included (see Section 8), the word SECTION is followed by a space, the priority-number, and a period. In either case, the remainder of the line is left blank.

A paragraph consists of one or more successive sentences, the first of which must be preceded by a paragraph-name. The paragraph-name starts at Margin A and is followed immediately by a period. A new paragraph is determined by the appearance of another paragraph-name.

All lines in a paragraph which follow the paragraph-name must start at Margin B. If a word or literal must be split over two lines, this will be indicated by placing a hyphen in the seventh character position of the second line. If the user prefers not to split a word or literal, he may leave the remainder of the line space filled and then start the word or literal on the next line.

8. SEGMENTATION

8.1. GENERAL

COBOL segmentation is the facility by which object program overlay can be accomplished through the compiler. This becomes necessary when an entire program cannot be contained in memory at one time. Through this feature, the less frequently-used routines need not be present in memory except when needed.

Because COBOL segmentation deals only with segmentation of procedures, not data, only the Procedure Division and the Environment Division are considered in determining segmentation requirements for an object program.

Proper use of segmentation is the responsibility of the programmer. The compiler has no way of determining its need or application independently. Segmentation is needed only when an entire program cannot be contained by the available memory. Normal practice is to restrict the number of overlay segments to a minimum and then only to infrequently used functions. In the extreme case, however, segmentation could be used to overlay one program with another almost completely by assigning only a small control function priority zero and assigning all other program functions different priority numbers.

8.2. ORGANIZATION

The Procedure Division of a source program is usually written in sections. Each section normally represents a set of closely related operations designed to perform a particular function. Normally, sectional division is not mandatory; however, when segmentation is used, the entire Procedure Division must be sectioned, and each section must be classified as either a ''fixed'' portion or an ''independent'', or ''overlay'' segment.

8.2.1. Fixed Portion

The fixed portion is that part of the object program which is logically treated as if it were always in memory. This portion of the program may be composed of two types of segments: *permanent segments* and *overlayable fixed segments*.

A permanent segment cannot be overlaid by any other part of the program. An overlayable fixed segment, although logically treated as if it were always in memory, can be overlaid, if necessary, by another segment to optimize memory utilization. However, such a segment, if called for by the program, is always made available in its last used state (see 8.4 for further explanation).

8.2.2. Independent Segment

An independent segment is a part of the object program which can overlay, and be overlaid by, either an overlayable fixed segment or another independent segment. It differs from an overlayable fixed segment in that it is always considered to be in its initial state each time it is made available to the program.

8.2.3. Segment Classification

Segment classification is accomplished by means of priority numbers in the section header. The priority is written as follows:

section-name SECTION [*priority-number*].

The priority-number must be an integer in the range 0 through 99. If omitted, a priority of zero (0) is assumed. Generally, the more often a segment is referred to, the lower its priority-number.

Sections which *must* always be available or which are referenced very frequently, will have a priority-number of 0. All segments with priority-number 0 through 49 belong to the fixed portion. Sections which frequently communicate with one another should be given the same priority-number.

Sections which are to be independent overlay segments will have priority-number 50 through 99. All sections that have the same particular priority number constitute a segment. Any two or more sections that communicate with each other must be members of the same segment (i.e., have the same priority number).

8.3. SEGMENTATION CONTROL

The logical sequence of the program is the same as the physical sequence. The compiler provides all necessary transfers of control from segment to segment, and provides control necessary for a segment to operate when that segment is called upon to be used. The segment under reference will be brought in before program continuation unless it is already in memory.

8.4. SEGMENT LIMIT

The memory area reserved for containing independent overlay segments is equal to the size of the largest overlay segment. However, when there is insufficient memory to contain all permanent segments plus the largest independent overlay segment, it is necessary to decrease the number of permanent segments. The SEGMENT-LIMIT feature provides the user with a method of reducing the number of permanent segments, while retaining the logical properties of fixed segments.

The SEGMENT-LIMIT clause appears in the OBJECT-COMPUTER paragraph and has the following format:

[, SEGMENT-LIMIT IS *priority-number*]

Priority number is an integer in the range 1 through 49. When the SEGMENT-LIMIT clause is specified, all fixed position segments with a priority number equal to, or greater than, the segment limit are thus defined as overlayable fixed segments; all fixed portion segments with priority number 0 up to, but not including, the segment limit constitute the permanent segment. All sections with priority-number 0 must always be part of the permanent segment. When the SEGMENT-LIMIT clause is not specified, all segments with priority numbers under 50 are permanent segments which cannot be overlaid.

8.4.1. Restrictions

When the SEGMENT-LIMIT feature is used, certain restrictions are placed on the use of the ALTER and PERFORM statements.

- ALTER Statement – A GO TO statement in any fixed segment (priority number 49 or less) can be ALTERed by an ALTER statement located in any other segment of the program. A GO TO statement in an overlayable or independent segment (priority number 50 or greater) can only be ALTERed by an ALTER statement located in the same segment as the GO TO statement. (Sections having the same priority number are considered to be in the same segment.)

- PERFORM Statement – The permissible range of a PERFORM statement is dependent upon whether the segment priority number is less than the SEGMENT-LIMIT, or equal to or greater than the SEGMENT-LIMIT.

  - Less than SEGMENT-LIMIT: The range of a PERFORM statement located in a fixed segment (priority number 49 or less) is restricted to the fixed portion of the segmented program. If the PERFORM statement is located in an overlayable or independent segment (priority number 50 or greater), its range is restricted to the segment in which it appears.

  - Equal to or greater than SEGMENT-LIMIT: If the PERFORM statement is located in a segment whose priority number is equal to or greater than the SEGMENT-LIMIT, its range is restricted to the segment in which it appears plus those segments having a priority number that is less than the SEGMENT -LIMIT.

    When a procedure-name in an overlayable or independent segment is referred to from within a PERFORM statement in a segment with a different priority number, the segment referred to is made available in its initial state for each iteration of the PERFORM statement.

The SEGMENT-LIMIT feature is not available on all compilers. If not available, all sections with priority numbers under 50 must remain in memory continuously at object time, since their effective priority-number would be zero (0).

9. THE COBOL LIBRARY

9.1. INTRODUCTION

The COBOL library contains entries available to a source program at compilation time through the use of the COPY statement. The effect of the compilation of library entries is the same as if the text were actually written as part of the source program.

The COBOL library may contain three types of entries as follows:

- Entries for the Environment Division consisting of equipment-oriented information.
- Entries for the Data Division consisting of information pertaining to file and data description entries.
- Entries for the Procedure Division consisting of sequences of procedure paragraphs and sections.

9.1.1. COPY

Format:

$$\underline{\text{COPY}}\ \textit{library-name}$$
$$\left[\begin{array}{l} \underline{\text{REPLACING}}\ \textit{word-1}\ \underline{\text{BY}} \begin{Bmatrix} \textit{word-2} \\ \textit{identifier-1} \\ \textit{literal-1} \end{Bmatrix} \\ \quad \left[,\ \textit{word-3}\ \underline{\text{BY}} \begin{Bmatrix} \textit{word-3} \\ \textit{identifier-2} \\ \textit{literal-2} \end{Bmatrix} \right] \ldots \end{array}\right]$$

Description:

The COPY statements permit the incorporation of existing library entries into the Environment, Data, and Procedure Divisions. A library entry is a segment of COBOL source language. By specifying the appropriate library-name within an appropriate format, the programmer can cause an entry to be copied from the library during compilation, and the result is the same as if the programmer had written the entry as part of his source program.

In this format, *word* represents any of the following:

data-name
procedure-name
condition-name
mnemonic-name
file-name

No other statement or clause may appear in the same entry as the COPY statement, and the library text to be COPYed must not contain any COPY statements. The COPYing process is terminated automatically when the end of the library text is reached. The COBOL implementor must specify whether the COPY statement itself or the statements of the library text to which it refers, or both, is to appear on the output listing. When both are to be listed, the relationship between them must be clearly indicated.

If the REPLACING option is used, each of the library words or identifiers specified in the format are replaced by the stipulated word or identifier with which it is associated in the format. This replacement does not alter the material as it exists in the library, and the entry may be called again in the same program with different replacements. Words specified in the REPLACING option may be any COBOL word except reserved words. The replacement of an identifier includes the replacement of all associated qualifiers, subscripts, and indexes.

9.2. LIBRARY ENTRIES FOR THE ENVIRONMENT DIVISION

There are five types of entries in the library that may be associated with the Environment Division: entries for the Special-Names, Source-Computer, Object-Computer, File-Control, and I-O-Control paragraphs.

To use an entry contained in the COBOL library, the COPY clause must follow the appropriate paragraph-name and indicate the statement (library-name) of the entry to be copied from the library. The formats are as follows:

SOURCE-COMPUTER. *copy-statement.*

OBJECT-COMPUTER. *copy-statement.*

SPECIAL-NAMES. *copy-statement.*

FILE-CONTROL. *copy-statement.*

I-O-CONTROL. *copy-statement.*

For example:

```
I-O-CONTROL. COPY LIB-IOC9.
```

If this entry is used, the library must contain an I-O-Control paragraph with the name LIB-IOC9.

9.3. LIBRARY ENTRIES FOR THE DATA DIVISION

Entries associated with the Data Division are of two types:

- Entries pertaining to the File Description portion of the Data Division.
- Entries pertaining to record descriptions in the Record Description or Working-Storage portions of the Data Division.

Each File Description paragraph described in the COBOL Library must have a level indicator associated with its first entry. This separates the information contained within the individual File Descriptions from adjacent File Descriptions.

This information may then be copied by use of the COPY option in the File Section of the Data Division. The Record Description entry associated with the FD entries is not copied from the library, however.

Data description entries differ from the File Description entries in that they are associated with level numbers rather than level indicators. The first entry in each set of contiguous data descriptions must carry level number 01; thus this set is separated from adjacent sets. A COPY clause causes all entries subordinate to the specified statement (library-name) to be copied from the library.

When copying a data description, the level number associated with the data description in the library must be the same as the entry containing the COPY clause in the source program.

In the File Section of the Data Division, the COPY clause takes the following formats:

FD *file-name copy-statement.*
01 *data-name copy-statement.*

For example, suppose that the following FD entry exists in the library:

FD EDITED-SHIPMENTS LABEL RECORD IS . . .

This entry could be called from the library by the COPY clause as follows:

FD MASTER-EDITED-SHIPMENTS COPY EDITED-SHIPMENTS.

Note that EDITED-SHIPMENTS is the statement (library-name) and MASTER-EDITED-SHIPMENTS is the file-name given in the Data Division of the source program. The FD entry would then be placed in the source program as follows:

FD MASTER-EDITED-SHIPMENTS LABEL RECORD IS . . .

Note also that the COPY clause copied only the File Description entry, not the associated Record Description entries.

The effect of the 01 data-name form of the COPY clause is to extract an entry or series of entries from the library and insert that information into the source program where the COPY clause appears. Thus the level-number and data-name are not replaced by the copied information.

For example:

01 SHIPMENTS-NOT-COSTED COPY DETAIL-SHIPMENTS.

In this case, the COBOL Library must contain a 01-level Record Description with the name DETAIL-SHIPMENTS. All data description entries from that point on are inserted into the source program in place of the COPY clause.

9.4. LIBRARY ENTRIES FOR THE PROCEDURE DIVISION

Each routine in the COBOL Library is composed of either one paragraph, identified by a paragraph-name, or one section, identified by a section-name. For purposes of copying this routine from the library, the paragraph-name or section-name is called a procedure-name.

Routines are retrieved from the library and copied into the source program through the use of the COPY statement in the Procedure Division. Then, at compile time the procedure-name that identifies the COPY statement replaces the library procedure-name that identifies the library routine, and all references to the routine procedure-name as it appears within the library routine are replaced by references to the procedure-name of the COPY statement. The format of a COPY clause written in the Procedure Division is as follows:

procedure-name. copy-statement.

APPENDIX A. RESERVED WORDS

The following is a list of reserved words used in COBOL. These words should not be used as names, or the results may be unpredictable.

ACCEPT
ACCESS
ACTUAL
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
AND
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
BEFORE
BEGINNING
BLANK
BLOCK
BY
CF
CH
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COMMA
COMP
COMPUTATIONAL
COMPUTE
CONFIGURATION
CONTAINS
CONTROL
CONTROLS
COPY
CORR
CORRESPONDING
CURRENCY
DATA
DATE-COMPILED
DATE-WRITTEN
DE
DECIMAL-POINT
DECLARATIVES
DEPENDING
DESCENDING
DETAIL
DISPLAY
DIVIDE
DIVISION
DOWN
ELSE
END
ENDING
ENTER
ENVIRONMENT
EQUAL
ERROR
EVERY
EXAMINE
EXIT
FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILLER
FINAL
FIRST
FOOTING
FOR
FROM
GENERATE
GIVING
GO
GREATER
GROUP
HEADING
HIGH-VALUE
HIGH-VALUES
HOLD
I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INITIATE
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
INVALID
IS
JUST
JUSTIFIED
KEY
KEYS
LABEL
LAST
LEADING
LEFT
LESS
LIMIT
LIMITS
LINE
LINE-COUNTER
LINES
LOCK
LOW-VALUE
LOW-VALUES
MEMORY
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY
NEGATIVE
NEXT
NO
NOT
NOTE
NUMBER
NUMERIC
OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
OUTPUT
PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE
PLUS
POSITION
POSITIVE
PROCEDURE
PROCEED
PROCESS
PROCESSING
PROGRAM-ID
QUOTE
QUOTES
RANDOM
RD
READ
RECORD
RECORDS
REDEFINES
REEL
RELEASE
REMARKS
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET

RETURN
REVERSED
REWIND
RF
RH
RIGHT
ROUNDED
RUN
SA
SAME
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENT-LIMIT

SELECT
SENTENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STATUS
STOP
SUBTRACT

SUM
SYNC
SYNCHRONIZED
TALLY
TALLYING
TAPE
TERMINATE
THAN
THROUGH } *equivalent*
THRU }
TIMES
TO
TYPE
UNIT
UNTIL
UP

UPON
USAGE
USE
USING
VALUE
VALUES
VARYING
WHEN
WITH
WORDS
WORKING-STORAGE
WRITE
ZERO
ZEROES
ZEROS

APPENDIX B. SUMMARY OF COBOL FORMATS

This appendix provides an alphabetical listing of the formats for the various statements and clauses available in the COBOL language. The list is in four parts, thus reflecting the four divisions of the COBOL source program. It is intended as a compact reference source to aid the programmer in observing specific rules of formatting. Since the COPY statement can appear in more than one division, see Section 9 for the COPY statement formats.

B1. IDENTIFICATION DIVISION.

B1.1. AUTHOR

[AUTHOR. [*comment-entry*]. . .]

B1.2. DATE-COMPILED

[DATE-COMPILED. [*comment-entry*]. . .]

B1.3. DATE-WRITTEN

[DATE-WRITTEN. [*comment-entry*]. . .]

B1.4. INSTALLATION

[INSTALLATION. [*comment-entry*]. . .]

B1.5. PROGRAM-ID

PROGRAM-ID. *program-name.*

B1.6. REMARKS

[REMARKS. [*comment-entry*]. . .]

B1.7. SECURITY

[SECURITY. [*comment-entry*]. . .]

B2. ENVIRONMENT DIVISION.

B2.1. FILE-CONTROL

```
FILE-CONTROL. { SELECT [OPTIONAL] file-name

ASSIGN TO [integer-1] implementor-name-1 [,implementor-name-2] . . .

                      [            { integer-2 }               [ { AREA  } ] ]   }
[FOR MULTIPLE REEL]   [ ,RESERVE   { NO        }  ALTERNATE    [ { AREAS } ] ] . } . . .
```

B2.2. I-O-CONTROL

```
I-O-CONTROL.

[ RERUN [ ON { file-name-1        } ]                          ]
[            { implementor-name   } ]                          ]
[                                                              ]
[         { { END OF REEL        }                 }           ] . . .
[         { { integer-1 RECORDS  } OF file-name-2  }           ]
[ EVERY   {   integer-2 CLOCK-UNITS                }           ]
[         {   condition-name                       }           ]

[; SAME [RECORD] AREA FOR file-name-3 {,file-name-4}. . .] . . .

[ ; MULTIPLE FILE TAPE CONTAINS file-name-5 [POSITION integer-3]

[,file-name-6 [POSITION integer-4] ] . . .] . . .      .
```

B2.3. OBJECT-COMPUTER

```
OBJECT-COMPUTER.  computer-name

[                       { WORDS      } ]
[ MEMORY SIZE integer   { CHARACTERS } ]  .
[                       { MODULES    } ]
```

B2.4. SOURCE-COMPUTER

```
SOURCE-COMPUTER.  computer-name.
```

B2.5. SPECIAL-NAMES

```
SPECIAL-NAMES.

[                   { IS mnemonic-name-1 [, ON STATUS IS condition-name-1
[ implementor-name  { IS mnemonic-name-2 [, OFF STATUS IS condition-name-3
[                   { ON STATUS IS condition-name-5
[                   { OFF STATUS IS condition-name-7

  [, OFF STATUS IS condition-name-2]] } ]
  [, ON STATUS IS condition-name-4]  ] } ] . . .
  [, OFF STATUS IS condition-name-6]   } ]
  [, ON STATUS IS condition-name-8]    } ]
  [, CURRENCY SIGN IS literal] [, DECIMAL-POINT IS COMMA].
```

B3. DATA DIVISION.

B3.1. BLANK

$$\left[;\ \underline{\text{BLANK}}\ \text{WHEN}\ \underline{\text{ZERO}}\right]$$

B3.2. BLOCK CONTAINS

$$\left[;\ \underline{\text{BLOCK}}\ \text{CONTAINS}\ [\textit{integer-1}\ \underline{\text{TO}}]\ \textit{integer-2}\left\{\begin{array}{l}\underline{\text{RECORDS}}\\ \text{CHARACTERS}\end{array}\right\}\right]$$

B3.3. DATA RECORDS

$$\left[;\ \underline{\text{DATA}}\left\{\begin{array}{l}\underline{\text{RECORDS}}\ \text{ARE}\\ \underline{\text{RECORD}}\ \text{IS}\end{array}\right\}\ \textit{data-name-1}\ [,\ \textit{data-name-2}]\ \ldots\right]$$

B3.4. JUSTIFIED

$$\left[;\left\{\begin{array}{l}\underline{\text{JUSTIFIED}}\\ \underline{\text{JUST}}\end{array}\right\}\ \underline{\text{RIGHT}}\right]$$

B3.5. LABEL RECORDS

$$;\ \underline{\text{LABEL}}\left\{\begin{array}{l}\underline{\text{RECORDS}}\ \text{ARE}\\ \underline{\text{RECORD}}\ \text{IS}\end{array}\right\}\left\{\begin{array}{l}\underline{\text{STANDARD}}\\ \underline{\text{OMITTED}}\\ \textit{data-name-1}\ [,\ \textit{data-name-2}]\ \ldots\end{array}\right\}$$

B3.6. LEVEL NUMBER

$$\textit{level-number}\left\{\begin{array}{l}\textit{data-name}\\ \underline{\text{FILLER}}\end{array}\right\}$$

B3.7. PICTURE

$$\left[;\left\{\begin{array}{l}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{array}\right\}\ \text{IS}\ \textit{character-string}\right]$$

B3.8. RECORD CONTAINS

$$\left[;\ \underline{\text{RECORD}}\ \text{CONTAINS}\ [\textit{integer-1}\ \underline{\text{TO}}]\ \textit{integer-2}\ \text{CHARACTERS}\right]$$

B3.9. REDEFINES

$$\textit{level-number}\ \textit{data-name-1}\ ;\ \underline{\text{REDEFINES}}\ \textit{data-name-2}$$

B3.10. RENAMES

$$66\ \textit{data-name-1};\ \underline{\text{RENAMES}}\ \textit{data-name-2}\ [\underline{\text{THRU}}\ \textit{data-name-3}]$$

B3.11. SYNCHRONIZED

$$\left[;\left\{\begin{array}{l}\underline{\text{SYNCHRONIZED}}\\ \underline{\text{SYNC}}\end{array}\right\}\left\{\begin{array}{l}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{array}\right\}\right]$$

B3.12. USAGE

$$\left[\ ;\ \text{USAGE IS}\ \left\{\begin{array}{l}\underline{\text{COMPUTATIONAL}}\\ \underline{\text{COMP}}\\ \underline{\text{DISPLAY}}\end{array}\right\}\ \right]$$

B3.13. VALUE IS

$$;\ \left\{\begin{array}{l}\underline{\text{VALUE}}\ \text{IS}\\ \underline{\text{VALUES}}\ \text{ARE}\end{array}\right\}\ \textit{literal-1}\ [\underline{\text{THRU}}\ \textit{literal-2}]\left[,\ \textit{literal-3}\ [\underline{\text{THRU}}\ \textit{literal-4}]\right]\ldots$$

B3.14. VALUE OF

$$\left[\begin{array}{l};\ \underline{\text{VALUE}}\ \underline{\text{OF}}\ \textit{data-name-1}\ \text{IS}\ \left\{\begin{array}{l}\textit{data-name-2}\\ \textit{literal-1}\end{array}\right\}\\ \qquad\qquad\left[\ ,\ \textit{data-name-3}\ \text{IS}\ \left\{\begin{array}{l}\textit{data-name-4}\\ \textit{literal-2}\end{array}\right\}\right]\ldots\end{array}\right]$$

B4. PROCEDURE DIVISION.

B4.1. ACCEPT

$$\underline{\text{ACCEPT}}\ \textit{identifier}\ [\underline{\text{FROM}}\ \textit{mnemonic-name}]$$

B4.2. ADD

Option 1:

$$\underline{\text{ADD}}\ \left\{\begin{array}{l}\textit{identifier-1}\\ \textit{literal-1}\end{array}\right\}\left[\ ,\left\{\begin{array}{l}\textit{identifier-2}\\ \textit{literal-2}\end{array}\right\}\right]\ldots$$

$$\underline{\text{TO}}\ \textit{identifier-m}\ [\underline{\text{ROUNDED}}]\left[,\ \textit{identifier-n}\ [\underline{\text{ROUNDED}}]\right]\ldots$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Option 2:

$$\underline{\text{ADD}}\ \left\{\begin{array}{l}\textit{identifier-1}\\ \textit{literal-1}\end{array}\right\},\left\{\begin{array}{l}\textit{identifier-2}\\ \textit{literal-2}\end{array}\right\}\left[\ ,\left\{\begin{array}{l}\textit{identifier-3}\\ \textit{literal-3}\end{array}\right\}\right]\ldots$$

$$\underline{\text{GIVING}}\ \textit{identifier-m}\ [\underline{\text{ROUNDED}}]$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Option 3:

$$\underline{\text{ADD}}\ \left\{\begin{array}{l}\underline{\text{CORRESPONDING}}\\ \underline{\text{CORR}}\end{array}\right\}\ \textit{identifier-1}\ \underline{\text{TO}}\ \textit{identifier-2}\ [\underline{\text{ROUNDED}}]$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

B4.3. ALTER

$$\underline{\text{ALTER}}\ \textit{procedure-name-1}\ \underline{\text{TO}}\ [\underline{\text{PROCEED}}\ \underline{\text{TO}}]\ \textit{procedure-name-2}$$

$$[\textit{procedure-name-3}\ \underline{\text{TO}}\ [\underline{\text{PROCEED}}\ \underline{\text{TO}}]\ \textit{procedure-name-4}]\ldots$$

B4.4. CLOSE

$$\underline{\text{CLOSE}}\ \textit{file-name-1}\ [\underline{\text{REEL}}]\left[\text{WITH}\left\{\begin{matrix}\underline{\text{NO}}\ \underline{\text{REWIND}}\\ \underline{\text{LOCK}}\end{matrix}\right\}\right]$$

$$\left[\ ,\ \textit{file-name-2}\qquad [\underline{\text{REEL}}]\left[\text{WITH}\left\{\begin{matrix}\underline{\text{NO}}\ \underline{\text{REWIND}}\\ \underline{\text{LOCK}}\end{matrix}\right\}\right]\right]\ldots$$

B4.5. COMPUTE

$$\underline{\text{COMPUTE}}\ \textit{identifier-1}\ [\underline{\text{ROUNDED}}]\ \underline{=}\ \left\{\begin{matrix}\textit{identifier-2}\\ \textit{literal}\\ \textit{arithmetic-expression}\end{matrix}\right\}$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

B4.6. DISPLAY

$$\underline{\text{DISPLAY}}\left\{\begin{matrix}\textit{literal-1}\\ \textit{identifier-1}\end{matrix}\right\}\left[\ ,\left\{\begin{matrix}\textit{literal-2}\\ \textit{identifier-2}\end{matrix}\right\}\right]\ldots$$

$$[\underline{\text{UPON}}\ \textit{mnemonic-name}]$$

B4.7. DIVIDE

Option 1:

$$\underline{\text{DIVIDE}} \left\{ \begin{matrix} \textit{identifier-1} \\ \textit{literal} \end{matrix} \right\} \ \underline{\text{INTO}}\ \textit{identifier-2}\ [\underline{\text{ROUNDED}}]$$

$$[\text{; ON }\underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Option 2:

$$\underline{\text{DIVIDE}} \left\{ \begin{matrix} \textit{identifier-1} \\ \textit{literal-1} \end{matrix} \right\} \underline{\text{INTO}} \left\{ \begin{matrix} \textit{identifier-2} \\ \textit{literal-2} \end{matrix} \right\} \ \underline{\text{GIVING}}\ \textit{identifier-3}\ [\underline{\text{ROUNDED}}]$$

$$[\text{; ON }\underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Option 3:

$$\underline{\text{DIVIDE}} \left\{ \begin{matrix} \textit{identifier-1} \\ \textit{literal-1} \end{matrix} \right\} \ \underline{\text{BY}} \left\{ \begin{matrix} \textit{identifier-2} \\ \textit{literal-2} \end{matrix} \right\} \underline{\text{GIVING}}\ \textit{identifier-3}\ [\underline{\text{ROUNDED}}]$$

$$[\text{; ON }\underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Option 4:

$$\underline{\text{DIVIDE}} \left\{ \begin{matrix} \textit{identifier-1} \\ \textit{literal-1} \end{matrix} \right\} \ \underline{\text{INTO}} \left\{ \begin{matrix} \textit{identifier-2} \\ \textit{literal-2} \end{matrix} \right\} \underline{\text{GIVING}}\ \textit{identifier-3}$$

$$[\underline{\text{ROUNDED}}]\ \underline{\text{REMAINDER}}\ \textit{identifier-4}$$

$$[\text{; ON }\underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Option 5:

$$\underline{\text{DIVIDE}} \left\{ \begin{matrix} \textit{identifier-1} \\ \textit{literal-1} \end{matrix} \right\} \ \underline{\text{BY}} \left\{ \begin{matrix} \textit{identifier-2} \\ \textit{literal-2} \end{matrix} \right\} \underline{\text{GIVING}}\ \textit{identifier-3}$$

$$[\underline{\text{ROUNDED}}]\ \underline{\text{REMAINDER}}\ \textit{identifier-4}$$

$$[\text{; ON }\underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

B4.8. ENTER

$$\underline{\text{ENTER}}\ \textit{language-name}\ [\textit{routine-name}].$$

B4.9. EXAMINE

Option 1:

$$\underline{\text{EXAMINE}}\ \textit{identifier}\ \underline{\text{TALLYING}} \left\{ \begin{array}{l} \underline{\text{ALL}} \\ \underline{\text{LEADING}} \\ \underline{\text{UNTIL}}\ \underline{\text{FIRST}} \end{array} \right\} \textit{literal-1}\ [\underline{\text{REPLACING}}\ \underline{\text{BY}}\ \textit{literal-2}]$$

Option 2:

$$\underline{\text{EXAMINE}}\ \textit{identifier}\ \underline{\text{REPLACING}} \left\{ \begin{array}{l} \underline{\text{ALL}} \\ \underline{\text{LEADING}} \\ [\underline{\text{UNTIL}}]\ \underline{\text{FIRST}} \end{array} \right\} \quad \textit{literal-1}\ \underline{\text{BY}}\ \textit{literal-2}$$

B4.10. EXIT

$$\underline{\text{EXIT}}.$$

B4.11. GO TO

Option 1:

$$\underline{\text{GO}}\ \underline{\text{TO}}\ [\textit{procedure-name}]$$

Option 2:

$$\underline{\text{GO}}\ \underline{\text{TO}}\ \textit{procedure-name-1}\ [,\ \textit{procedure-name-2}]\ \ldots,\ \textit{procedure-name-n}$$

$$\underline{\text{DEPENDING}}\ \text{ON}\ \textit{identifier}$$

B4.12. MOVE

Option 1:

$$\underline{\text{MOVE}} \left\{ \begin{array}{l} \textit{identifier-1} \\ \textit{literal} \end{array} \right\}\ \underline{\text{TO}} \left\{ \textit{identifier-2} \right\} \ldots$$

Option 2:

$$\underline{\text{MOVE}} \left\{ \begin{array}{l} \underline{\text{CORRESPONDING}} \\ \underline{\text{CORR}} \end{array} \right\} \textit{identifier-1}\ \underline{\text{TO}}\ \textit{identifier-2}$$

B4.13. MULTIPLY

Option 1:

$$\underline{\text{MULTIPLY}} \left\{ \begin{matrix} \textit{identifier-1} \\ \textit{literal} \end{matrix} \right\} \underline{\text{BY}}\ \textit{identifier-2}\ [\underline{\text{ROUNDED}}]$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Option 2:

$$\underline{\text{MULTIPLY}} \left\{ \begin{matrix} \textit{identifier-1} \\ \textit{literal-1} \end{matrix} \right\} \underline{\text{BY}} \left\{ \begin{matrix} \textit{identifier-2} \\ \textit{literal-2} \end{matrix} \right\}$$

$$\underline{\text{GIVING}}\ \textit{identifier-3}\ [\underline{\text{ROUNDED}}]$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

B4.14. NOTE

$$\underline{\text{NOTE}}\ \textit{character-string.}$$

B4.15. OPEN

$$\underline{\text{OPEN}} \left[\underline{\text{INPUT}} \left\{ \textit{file-name} \left[\begin{matrix} \underline{\text{REVERSED}} \\ \text{WITH}\ \underline{\text{NO}}\ \underline{\text{REWIND}} \end{matrix} \right] \right\} \ldots \right]$$

$$\left[\underline{\text{OUTPUT}} \left\{ \textit{file-name}\ [\text{WITH}\ \underline{\text{NO}}\ \underline{\text{REWIND}}] \right\} \ldots \right]$$

B4.16. PERFORM

Option 1:

$$\underline{\text{PERFORM}}\ \textit{procedure-name-1}\ [\underline{\text{THRU}}\ \textit{procedure-name-2}]$$

Option 2:

$$\underline{\text{PERFORM}}\ \textit{procedure-name-1}\ [\underline{\text{THRU}}\ \textit{procedure-name-2}] \left\{ \begin{matrix} \textit{identifier} \\ \textit{integer} \end{matrix} \right\} \underline{\text{TIMES}}$$

Option 3:

$$\underline{\text{PERFORM}}\ \textit{procedure-name-1}\ [\underline{\text{THRU}}\ \textit{procedure-name-2}]\ \underline{\text{UNTIL}}\ \textit{condition}$$

Option 4:

$$\underline{\text{PERFORM}}\ \textit{procedure-name-1}\ [\underline{\text{THRU}}\ \textit{procedure-name-2}]$$

$$\underline{\text{VARYING}}\ \textit{identifier-1}\ \underline{\text{FROM}} \left\{ \begin{matrix} \textit{literal-1} \\ \textit{identifier-2} \end{matrix} \right\}$$

$$\underline{\text{BY}} \left\{ \begin{matrix} \textit{literal-2} \\ \textit{identifier-3} \end{matrix} \right\} \underline{\text{UNTIL}}\ \textit{condition}$$

B4.17. READ

$$\underline{\text{READ}}\ \textit{file-name}\ \text{RECORD}\ [\underline{\text{INTO}}\ \textit{identifier}]$$

$$;\ \text{AT}\ \underline{\text{END}}\ \textit{imperative-statement}$$

B4.18. STOP

$$\underline{\text{STOP}}\left\{\begin{matrix}\textit{literal}\\ \underline{\text{RUN}}\end{matrix}\right\}$$

B4.19. SUBTRACT

Option 1:

$$\underline{\text{SUBTRACT}}\left\{\begin{matrix}\textit{literal-1}\\ \textit{identifier-1}\end{matrix}\right\}\left[,\left\{\begin{matrix}\textit{literal-2}\\ \textit{identifier-2}\end{matrix}\right\}\right]\ldots$$

$$\underline{\text{FROM}}\ \textit{identifier-m}\ [\underline{\text{ROUNDED}}]\left[,\ \textit{identifier-n}\ [\underline{\text{ROUNDED}}]\right]\ldots$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Option 2:

$$\underline{\text{SUBTRACT}}\left\{\begin{matrix}\textit{literal-1}\\ \textit{identifier-1}\end{matrix}\right\}\left[,\left\{\begin{matrix}\textit{literal-2}\\ \textit{identifier-2}\end{matrix}\right\}\right]\ldots\ \underline{\text{FROM}}\left\{\begin{matrix}\textit{literal-m}\\ \textit{identifier-m}\end{matrix}\right\}$$

$$\underline{\text{GIVING}}\ \textit{identifier-n}\ [\underline{\text{ROUNDED}}]$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

Option 3:

$$\underline{\text{SUBTRACT}}\left\{\begin{matrix}\underline{\text{CORRESPONDING}}\\ \underline{\text{CORR}}\end{matrix}\right\}\textit{identifier-1}\ \underline{\text{FROM}}\ \textit{identifier-2}\ [\underline{\text{ROUNDED}}]$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

B4.20. WRITE

$$\underline{\text{WRITE}}\ \textit{record-name}\ [\underline{\text{FROM}}\ \textit{identifier-1}]$$

$$\left[\left\{\begin{matrix}\underline{\text{BEFORE}}\\ \underline{\text{AFTER}}\end{matrix}\right\}\text{ ADVANCING }\left\{\begin{matrix}\textit{identifier-2}\ \text{LINES}\\ \textit{integer}\ \text{LINES}\\ \textit{mnemonic-name}\end{matrix}\right\}\right]$$

APPENDIX C. SAMPLE PROBLEM

The sample inventory update problem introduced in Section 2 and developed throughout the manual is shown in this section in its entirety.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. MASTER-INVENTORY-UPDATE.
000300 AUTHOR. AB. C, DEFGHI.
000350 INSTALLATION. OSHKOSH.
000400 DATE-WRITTEN. JUNE 7 1966.
000450 DATE-COMPILED. JUNE 23 1966.
000500 SECURITY. CLASS B COMPANY CONFIDENTIAL.
000600 REMARKS. MASTER INVENTORY ON TAPE IS RUN AGAINST DETAIL
000700     TRANSACTIONS FOR UPDATING TO CREATE A NEW MASTER
000800     INVENTORY FILE. NEW STOCK ITEMS MAY BE ADDED BY
000900     ZERO-TYPE DETAIL TRANSACTIONS.
001000     STOCK ITEMS ON WHICH THE QUANTITY ON HAND HAS
001100     FALLEN BELOW REORDER LEVEL WILL BE LISTED WITH
001200     PURCHASING INFORMATION ON THE PRINTER, AS WILL
001300     BE ALL NEW STOCK ITEMS.
001400 ENVIRONMENT DIVISION.
001500 CONFIGURATION SECTION.
001600 SOURCE-COMPUTER. MARK-I.
001700 OBJECT-COMPUTER. MARK-I.
001800     MEMORY SIZE 20000 WORDS.
001850 SPECIAL-NAMES. NEW-PAGE IS TOP-OF-NEXT-PAGE.
002100 INPUT-OUTPUT SECTION.
002200 FILE-CONTROL.
002300     SELECT OLD-MASTER-INVENTORY ASSIGN TO MAGNETIC-TAPE-UNIT.
002400     SELECT NEW-MASTER-INVENTORY ASSIGN TO MAGNETIC-TAPE-UNIT.
002600     SELECT DETAIL-TRANSACTION-CARDS ASSIGN TO CARD-READER.
002610     SELECT REORDER-LIST ASSIGN TO PRINTER.
002640 I-O-CONTROL.
002650 RERUN EVERY END OF REEL OF NEW-MASTER-INVENTORY.
003000 DATA DIVISION.
003100 FILE SECTION.
003200 FD  OLD-MASTER-INVENTORY
003210     LABEL RECORD IS STANDARD
003250     VALUE OF ID IS 'MSTINVP'
003300     BLOCK CONTAINS 50 RECORDS
003400     DATA RECORD IS MASTER-RECORD.
003500 01  MASTER-RECORD.
003600     03 SEQ-STOCK-NUMBER PICTURE IS X(6).
003700     03 NUMBER-MANUFACTURER PICTURE IS 9(3).
003900     03 MFR-CATALOG-NUMBER PICTURE IS X(10).
004000     03 DESCRIPTION PICTURE IS X(30).
004100     03 ON-HAND-UNITS PICTURE IS 9(4) USAGE IS COMPUTATIONAL.
004300     03 COST-PER-UNIT PICTURE IS 9(4)V99.
004400     03 TOTAL-WHOLESALE-VALUE PICTURE IS 9(8)V99.
004500     03 MIN-STOCK-UNIT-QUANTITY PICTURE IS 9(4).
004700 FD  DETAIL-TRANSACTION-CARDS DATA RECORD IS TRANSACTIONS
004800     LABEL RECORDS OMITTED.
```

```
005000 01  TRANSACTIONS.
005100     05 STOCK-CONTROL-NUMBER PICTURE IS X(6).
005110     05 NO-TRANSACTOR PICTURE IS 999.
005120     05 ORDER-NUMBER PICTURE IS X(10).
005200     05 DTL-DESCRIPTION PICTURE IS X(30).
005300     05 TYPE-TRANSACTION PICTURE IS 9 USAGE IS COMPUTATIONAL.
005500     05 QUANTITY PICTURE IS 9(4).
005600     05 UNIT-COST PICTURE IS 9(4)V99.
006010     05 FILLER PICTURE IS X(20).
006100 FD  NEW-MASTER-INVENTORY
006110     LABEL RECORD IS STANDARD
006120     BLOCK CONTAINS 50 RECORDS
006130     VALUE OF ID IS 'MSTINVP'
006140     DATA RECORD IS MASTER-RECORD.
006150 01  MASTER-RECORD.
006160     04 SEQ-STOCK-NUMBER PICTURE X(6).
006170     04 NUMBER-MANUFACTURER PICTURE IS 9(3).
006180     04 MFR-CATALOG-NUMBER PICTURE IS X(10).
006190     04 DESCRIPTION PICTURE IS X(30).
006200     04 ON-HAND-UNITS PICTURE IS 9(4) USAGE IS COMPUTATIONAL.
006210     04 COST-PER-UNIT PICTURE IS 9(4)V99.
006220     04 TOTAL-WHOLESALE-VALUE PICTURE 9(8)V99.
006230     04 MIN-STOCK-UNIT-QUANTITY PICTURE IS 9(4).
006500 FD  REORDER-LIST LABEL RECORD IS OMITTED
006600     DATA RECORD IS REPLENISH-STOCK-ITEM.
006700 01  REPLENISH-STOCK-ITEM.
006750     08 FILLER PICTURE IS X(5).
006800     08 STK-NUMBER-PRINT PICTURE IS X(6).
006900     08 FILLER PICTURE IS X(5).
007000     08 NO-MFR PICTURE IS 9(3).
007100     08 FILLER PICTURE IS X(7).
007250     08 FLAG-NEW-STOCK PICTURE IS A.
007300     08 FILLER PICTURE IS X(7).
007350     08 MFR-ORDER-NUMBER PICTURE IS X(10).
007400     08 FILLER PICTURE IS X(7).
007450     08 ITEM-DESCRIPTION PICTURE IS X(30).
007500     08 FILLER PICTURE IS X(5).
007550     08 UNITS-ON-HAND PICTURE IS ZZZ9.
007575     08 FILLER PICTURE IS X(5).
007600     08 EMERGENCY-REORDER-FLAG PICTURE IS X(3).
007700     08 FILLER PICTURE IS X(5).
007800     08 MIN-UNITS PICTURE IS ZZZ9.
007900     08 FILLER PICTURE IS X(5).
008000     08 UNIT-COST PICTURE IS $$$$Z.99.
008100     08 FILLER PICTURE IS X(12).
008200 WORKING-STORAGE SECTION.
008300 77  DIVIDEND PICTURE IS 9(6) VALUE IS ZERO.
008400 77  PERCENTAGE PICTURE IS 9999 VALUE IS 0000.
008500 77  SWITCH PICTURE IS 9 VALUE IS ZERO USAGE IS COMPUTATIONAL.
008610 77  LINE-NO VALUE IS 0 PICTURE IS 99.
```

```
008700 01  1-REPLENISH-REPORT-HEADING.
008800     10 FILLER PICTURE IS X(116) VALUE SPACES.
008900     10 PAGE.
009100     15 PAGEKON VALUE IS 'PAGE' PICTURE IS X(5).
009110     15 PAGE-NO PICTURE IS 999 VALUE ZERO.
009150     15 FILLER PICTURE IS X(8) VALUE SPACES.
009300 01  2-REPLENISH-REPORT-HEADING.
009350     02 ONE-THRU-SIXTY-ONE PICTURE X(61) VALUE IS '     STOCK
009400-    ' MFR      NEW       CATALOG              ITE'.
009450     02 SIXTY-TWO-THRU-132 PICTURE X(71) VALUE IS 'M - UNIT
009500-    '            UNITS   EMERG     MIN         UNIT            '.
009700 01  3-REPLENISH-REPORT-HEADING.
009750     02 ONE-THRU-71 PICTURE X(71) VALUE IS '    NUMBER     NUMBER
009800-    '  STOCK      NUMBER              DESCRIPTION'.
009900     02 SEVENTY-ONE-THRU-132 PICTURE X(61) VALUE
010000     '              ON HAND  REORD    UNITS       COST'.
010200 PROCEDURE DIVISION.
010300 INITIALIZE.
010400     OPEN INPUT OLD-MASTER-INVENTORY, DETAIL-TRANSACTION-CARDS.
010500     OPEN OUTPUT NEW-MASTER-INVENTORY, REORDER-LIST.
010700     PERFORM HEADER-PRINT.
010800 GET-NEXT-TRANSACTION.
010900     READ DETAIL-TRANSACTION-CARDS AT END GO TO SET-UP-END-RUN.
020000 READ-INVENTORY-RECORD.
020100     IF SWITCH IS EQUAL TO 1 GO TO RESET-SWITCH.
020200     READ OLD-MASTER-INVENTORY INTO MASTER-RECORD OF
020300     NEW-MASTER-INVENTORY AT END GO TO CLOSE-FILES.
020800 RESET-SWITCH.
020900     MOVE ZERO TO SWITCH.
030000 CHECK-STOCK-NUMBER.
030100     IF STOCK-CONTROL-NUMBER GREATER THAN SEQ-STOCK-NUMBER
030200     OF NEW-MASTER-INVENTORY GO TO FINISH-MASTER.
030300     MOVE 1 TO SWITCH.
030400     IF STOCK-CONTROL-NUMBER IS EQUAL SEQ-STOCK-NUMBER
030500     OF NEW-MASTER-INVENTORY GO TO UPDATE.
030900     IF TYPE-TRANSACTION IS EQUAL TO ZERO GO TO NEW-STOCK-ITEM.
040000 ERROR-MESSAGE.
040100     DISPLAY STOCK-CONTROL-NUMBER ' NOT IN FILE, TRANSACTION
040200-    ' TYPE IS  ' TYPE-TRANSACTION.
040300     GO TO GET-NEXT-TRANSACTION.
040400 UPDATE.
040600     IF TYPE-TRANSACTION IS EQUAL TO 1 OR 2 GO TO SHIPMENT.
040700     IF TYPE-TRANSACTION IS EQUAL TO 3 OR 4 GO TO RECEIPT.
040800     GO TO ERROR-MESSAGE.
050300 SHIPMENT.
050400     SUBTRACT QUANTITY FROM ON-HAND-UNITS
050500     IN NEW-MASTER-INVENTORY.
050600     GO TO GET-NEXT-TRANSACTION.
050700 RECEIPT.
050800     ADD QUANTITY TO ON-HAND-UNITS IN
050900     NEW-MASTER-INVENTORY.
060000     GO TO GET-NEXT-TRANSACTION.
```

```
060100 FINISH-MASTER.
060200     MULTIPLY ON-HAND-UNITS OF NEW-MASTER-INVENTORY BY COST-PER-UN
060300-    IT OF NEW-MASTER-INVENTORY GIVING TOTAL-WHOLESALE-VALUE OF
060400     NEW-MASTER-INVENTORY ON SIZE ERROR DISPLAY 'OVERFLOW ON TOTAL
060410-    ' WHOLESALE VALUE'.
060420     DISPLAY SEQ-STOCK-NUMBER OF MASTER-RECORD IN
060430     NEW-MASTER-INVENTORY.
060500     IF FLAG-NEW-STOCK IS EQUAL TO 'N' MOVE 1 TO SWITCH.
060600     IF MIN-STOCK-UNIT-QUANTITY OF NEW-MASTER-INVENTORY IS GREATER
060700     THAN ON-HAND-UNITS IN NEW-MASTER-INVENTORY OR FLAG-NEW-STOCK
060800     IS EQUAL TO 'N' PERFORM REORDER-STOCK.
070100     PERFORM WRITE-NEW-INVENTORY.
070200     MOVE MASTER-RECORD OF OLD-MASTER-INVENTORY TO MASTER-RECORD
070210     OF NEW-MASTER-INVENTORY.
070300     GO TO READ-INVENTORY-RECORD.
070400 WRITE-NEW-INVENTORY.
070500     WRITE MASTER-RECORD OF NEW-MASTER-INVENTORY.
070700 REORDER-STOCK.
080000     MULTIPLY ON-HAND-UNITS OF NEW-MASTER-INVENTORY BY 100
080050     GIVING DIVIDEND ON SIZE ERROR DISPLAY
080060     'OVERFLOW ON MULTIPLICATION'.
080070     DISPLAY SEQ-STOCK-NUMBER OF MASTER-RECORD IN
080080     NEW-MASTER-INVENTORY.
080100     DIVIDE MIN-STOCK-UNIT-QUANTITY OF NEW-MASTER-INVENTORY INTO
080200     DIVIDEND GIVING PERCENTAGE ROUNDED.
080300     IF PERCENTAGE IS LESS THAN 60 MOVE 'EEE' TO
080400     EMERGENCY-REORDER-FLAG.
080500     MOVE SEQ-STOCK-NUMBER OF NEW-MASTER-INVENTORY TO
080600     STK-NUMBER-PRINT.
080700     MOVE NUMBER-MANUFACTURER OF NEW-MASTER-INVENTORY TO NO-MFR.
080900     MOVE MFR-CATALOG-NUMBER OF NEW-MASTER-INVENTORY TO
090000     MFR-ORDER-NUMBER OF REORDER-LIST.
090100     MOVE DESCRIPTION IN NEW-MASTER-INVENTORY TO ITEM-DESCRIPTION.
090200     MOVE ON-HAND-UNITS IN NEW-MASTER-INVENTORY TO UNITS-ON-HAND.
090400     MOVE MIN-STOCK-UNIT-QUANTITY OF NEW-MASTER-INVENTORY TO
090500     MIN-UNITS.
090600     MOVE COST-PER-UNIT IN NEW-MASTER-INVENTORY TO UNIT-COST
090610     OF REORDER-LIST.
090700     WRITE REPLENISH-STOCK-ITEM BEFORE ADVANCING 3 LINES.
090800     ADD 1 TO LINE-NO.
090810     MOVE SPACES TO REPLENISH-STOCK-ITEM.
090900     IF LINE-NO IS GREATER THAN 17 PERFORM HEADER-PRINT.
100100 NEW-STOCK-ITEM.
100500     MOVE SPACES TO MASTER-RECORD OF NEW-MASTER-INVENTORY.
100600     MOVE STOCK-CONTROL-NUMBER TO SEQ-STOCK-NUMBER IN
100700     NEW-MASTER-INVENTORY.
100800     MOVE NO-TRANSACTOR TO NUMBER-MANUFACTURER IN
100900     NEW-MASTER-INVENTORY.
200200     MOVE ORDER-NUMBER TO MFR-CATALOG-NUMBER IN
200300     NEW-MASTER-INVENTORY.
```

```
200400     MOVE DTL-DESCRIPTION TO DESCRIPTION IN NEW-MASTER-INVENTORY.
200600     MOVE ZEROS TO ON-HAND-UNITS OF NEW-MASTER-INVENTORY,
200700     TOTAL-WHOLESALE-VALUE IN NEW-MASTER-INVENTORY.
200800     MOVE UNIT-COST OF TRANSACTIONS TO COST-PER-UNIT IN
200900     NEW-MASTER-INVENTORY.
300000     MOVE QUANTITY TO MIN-STOCK-UNIT-QUANTITY IN
300010     NEW-MASTER-INVENTORY.
300100     MOVE 'N' TO FLAG-NEW-STOCK.
300200     GO TO GET-NEXT-TRANSACTION.
300300 HEADER-PRINT.
300400     ADD 1 TO PAGE-NO.
300500     WRITE REPLENISH-STOCK-ITEM FROM 1-REPLENISH-REPORT-HEADING
300510     AFTER ADVANCING NEW-PAGE LINES.
300600     WRITE REPLENISH-STOCK-ITEM FROM 2-REPLENISH-REPORT-HEADING.
300610     WRITE REPLENISH-STOCK-ITEM FROM 3-REPLENISH-REPORT-HEADING.
300620     MOVE SPACES TO REPLENISH-STOCK-ITEM.
300630     WRITE REPLENISH-STOCK-ITEM.
300700     MOVE ZEROS TO LINE-NO.
300900 SET-UP-END-RUN.
400000     MOVE '......' TO STOCK-CONTROL-NUMBER.
400100     GO TO READ-INVENTORY-RECORD.
400200 CLOSE-FILES.
400300     CLOSE OLD-MASTER-INVENTORY, DETAIL-TRANSACTION-CARDS.
400690     CLOSE NEW-MASTER-INVENTORY, REORDER-LIST.
400700     STOP RUN.
```

INDEX

BOOK FOLDER SIDE 1 FRONT