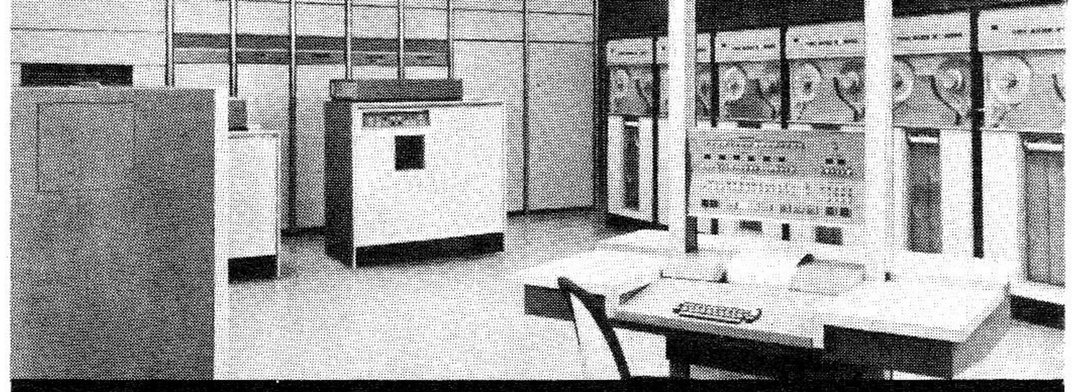
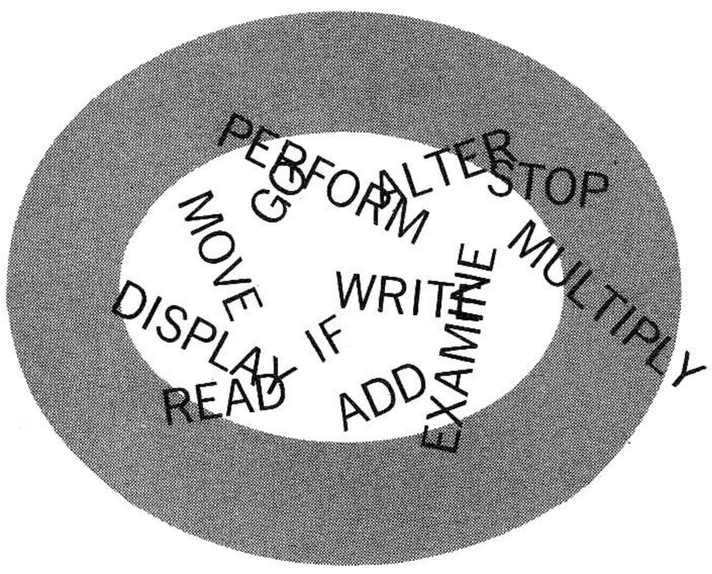


UNIVAC® 1107

TECHNICAL BULLETIN

COBOL

Programmer's Guide



U-2582

INDEX

PAGE

PAGE

UNIVAC 1107
COBOL, U-2582

June 1963

UPDATING PACKAGE A

The attached sheets contain corrections to the COBOL Programmer's Guide. No new sections or major revisions are included in this updating package.

CONTENTS:

All pages listed above should replace the corresponding pages in the subject manual. Some of the pages do not contain revisions, but are published in this package because they are the front or back of pages which have been changed.

It is suggested that this summary sheet be retained in the manual to serve as a reference or catalog of changes.

I. INTRODUCTION

1. OBJECTIVES

There are hundreds of business, government, and educational organizations using a wide variety of electronic computers in data processing operations. Some of the major users have more than one type of computer applied to the same general data processing application at different locations. The experience of these organizations to date indicates that a major problem in using computing equipment wisely and efficiently lies in stating the data processing application in such a way that computer programs are developed and maintained with a minimum of time and programming effort.

A COmmon Business Oriented Language, independent of any make or model of computer, open-ended and stated in English, would do much to solve or reduce this problem. Such a language would also simplify and speed up the related problem of training personnel in the design of data processing systems and the development of computer programs for such systems.

In general, the following situations represent some of the areas requiring the development of a COmmon Business Oriented Language for programming computers:

a. The need to develop data processing systems for existing computers that can be processed on future, more powerful computers with a minimum of conversion costs. For any user possessing computers of different manufacture and therefore not compatible, this need exists for efficient translation of a data system from one computer type to another.

b. With the rapidly changing and expanding requirements of management, data processing systems need constant revision and augmentation. Full documentation of the present system is required in a form conducive to making such changes and additions with minimum time and costs.

c. Many users are faced with the need to produce a large number of computer programs in a short period of time. This places a heavy burden on the existing programming staff or requires quick augmentation with relatively inexperienced programmers.

2. HISTORY

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 type 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 this time. The concept of three committees or task forces was agreed upon. (The terms "committee", "task force", and "group" are used interchangeably in this report.) 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 in 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 latter two groups were used 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 polish the system by December 1959. It was 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", which suggests 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, 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 changed to keep COBOL up-to-date. This Maintenance Committee comprised Users and Manufacturers Groups. The Maintenance Committee considered and agreed on a number of revisions to COBOL.

In order to devote concentrated attention to bringing out a revised and up-dated "COBOL-1961", a Special Task Group was created by the Executive Committee. The attached "COBOL-1961" Manual reflects the work of this Special Task Group.

Members of this Special Task Group represented the following organizations:

Air Material Command, United States Air Force
Bendix Corporation, Computer Division
Burroughs Corporation
Control Data Corporation
David Taylor Model Basin, Bureau of Ships
DuPont Company
General Electric Company
International Business Machines Corporation
Minneapolis-Honeywell Regulator Company
National Cash Register Company
Philco Corporation
Radio Corporation of America
Sylvania Electric Products, Inc.
Univac Division of Sperry Rand Corporation
U. S. Steel Corporation

The sessions of the Special Task Group were chaired by representatives of using organizations, specifically the Air Materiel Command and DuPont Company.

3. PHASING

The following three paragraphs are here included as the official position of the CODASYL Committee in their publication of May 5, 1961.

It is presently planned by the Executive Committee to make annnual revisions to the COBOL Manual. Accordingly, it is expected that this COBOL-61 Manual will be superceded by COBOL-62 to be released early in the calendar year 1962.

The prospective user of COBOL is reminded that a source program language such as COBOL must be dynamic to be effective, and will be subject to continuous change. Computer manufacturers should advise prospective users very specifically as to what features of COBOL are provided for in compilers at any particular point in time. Implementation of all of the "REQUIRED" elements is expected of each manufacturer within the limits of hardware capability. With respect to "ELECTIVE" elements (as defined below), although implementation is optional, if a manufacturer does choose to implement, he is expected to implement (within the limits of hardware capability) in accordance with the specifications for that element contained in the manual.

The category of "ELECTIVE COBOL-61" was established as a temporary expedient only, to permit manufacturers to release a "proper" COBOL compiler while still in the process of development on some specifically designated features in the language.

Definition of "REQUIRED COBOL - 1961"

"REQUIRED COBOL - 1961" consists of that group of features and options, within the complete COBOL specifications for the year 1961, which have been designated as comprising the minimum subset of the total language which must be implemented (to the extent of hardware capability) by any implementor claiming a "proper" COBOL - 1961 compiler.

The Executive Committee recommends that each user of COBOL-61 recognize his own responsibility to determine that the compiler offered by any manufacturer does in fact implement all the "REQUIRED" features to the extent of hardware capability to make it a "proper" COBOL compiler.

Definition of "ELECTIVE COBOL - 1961"

"ELECTIVE COBOL - 1961" consists of those features and options, within the complete COBOL specifications for the year 1961, whose implementation has been designated as optional for the manufacturers for the year 1961. If an implementor chooses to include any of these features or options (either totally or partially) in his compiler for 1961, he is expected to implement them in accordance with the specifications, for the feature or option, which are given in the COBOL-61 Manual.

4. MAINTENANCE

In recognition of the fact that the task of defining a COmmon Business Oriented Language does not end with publishing specifications, the Executive Committee has created a Maintenance Committee.

The Maintenance Committee is comprised of a Users Group and a Manufacturer Group. Its task is to give continuing attention to the system in order to answer questions arising from users and implementors of the language, and also to make definitive modifications (including additions, clarifications and changes). Additions, clarifications, and changes to COBOL, on which the Users and Manufacturers Groups are agreed, will be reproduced as working papers pending the next annual publication of a revised COBOL Manual. Proposals for supplements to COBOL will be accepted from outside organizations or individuals by the Executive Committee, and sent to the Users and Manufacturers Groups for concurrent consideration.

5. ACKNOWLEDGMENT

The Executive Committee has requested that all organizations who intend to implement the COBOL system, and expect to write a manual describing the operation of their processor of the COBOL system, reproduce the remainder of the Acknowledgment Section in its entirety as part of the preface to any such publication.

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

Air Material Command, United States Air Force
Bureau of Standards, Department of Commerce
David Taylor Model Basin, Bureau of Ships, U. S. Navy
Electronic Data Processing Division, Minneapolis-Honeywell Regulator Company

Burroughs Corporation
International Business Machines Corporation
Radio Corporation of America
Sylvania Electric Products, Inc.
Univac Division of Sperry Rand Corporation

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

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

This COBOL-61 manual is the result of contributions made by all of the above-mentioned organizations. No warranty, expressed or implied, is made by any contributor or by the committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

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

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

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

II. INTRODUCTION TO THE COBOL LANGUAGE

1. GENERAL

Perhaps the best way to understand how the COBOL system works is this: The COBOL language contains a basic list of key words and symbols. Each key word and symbol specifies to the processor a definite set of machine operations. In effect, the programmer thus has at his disposal a whole series of "prefabricated" portions of the object program he wishes the computer to construct. When he writes a COBOL-language program, he is actually directing the computer to bring together, in the proper sequence, the groups of machine instructions necessary to accomplish the desired result. The language in which he does this is not only easy to work with; it saves him from having to specify a great many machine steps in detail. The rules for writing the COBOL language are much simpler than those which govern the machine languages, and the programmer is enabled to write his programs easily, rapidly, and accurately. He can use English words to direct and control the complicated operations of the computer.

2. SIMPLIFIED NARRATIVE PROBLEM

a. General

The COBOL language facilitates communication between business English and the language of the computer, using English words and certain rules of syntactical English. Because COBOL is a computer language, it must be a very precise language and strict adherence to the COBOL rules is mandatory.

A simplified problem on inventory control follows, presenting an overall view of this problem expressed in the COBOL language.

b. Definition of Problem

The problem is an inventory updating function involving processing of detail transactions; i.e., shipments or receipts, against a master inventory file. This necessitates changing quantity on hand, total dollar value and checking balance against a predetermined minimum stock level.

c. Identification of Data

In the interest of simplification, the item description will not be complete but restricted to data-name only.

The problem uses two input files and two output files. One of the inputs, the master-record, is also one of the outputs, the updated-master-record.

The files, input or output, consist of a series of individual records. All records on each file are equal size and are assured to have similar contents. Therefore, the description of one record describes all records.

d. Organization of Data

The source for the master inventory file possibly might be a ledger card similar to the one below:

INVENTORY LEDGER

Stock Number	Description	Quantity on Hand	Unit Cost	Total $ Value	Reorder Level
A 1234	Fishing Rod, Casting	0100	04.50	00045000	0075
A 3456	Fishing Rod, Deep Sea	0095	09.50	00090250	0050
B 7890	Reel, Spinning	0200	08.00	00160000	0100
B 8901	Reel, Casting	0065	07.00	00045500	0050

FIGURE II-1

Each line in Figure II-1 is a complete item record and the fields are of equal size, therefore one record definition would provide the definition for all records on the inventory ledger.

In the problem, we shall refer to the inventory ledger records as INVENTORY-MASTER and we shall assign data names to each field as follows:

STOCK-NUMBER
DESCRIPTION
QUANTITY-ON-HAND
UNIT-COST
TOTAL-DOLLAR-VALUE
REORDER-LEVEL

Note: This is not a complete item description entry for COBOL.

The detail transactions are used to introduce changes to the INVENTORY-MASTER and they must be introduced in the same sequence. They must also have an identical field for matching with the INVENTORY-MASTER.

A sample detail transaction might appear as follows:

DETAIL TRANSACTION

Stock Number	Description	Shipments	Receipts
A 3456	Fishing Rod, Deep Sea	045	025

FIGURE II-2

We shall call the detail transaction DETAIL and assign data names to each field as follows:

STOCK-NUMBER
DESCRIPTION
SHIPMENTS
RECEIPTS

Note: Again, this is not a complete item description.

The identical field used for matching would be the STOCK-NUMBER field of both the INVENTORY-MASTER and the DETAIL.

Note: Both these records are using the same data-name for the field and the computer must have some method of determining which record is being addressed, therefore the STOCK-NUMBER field of each record must be made unique by qualifying it with the record name; i. e.,

STOCK-NUMBER IN DETAIL
STOCK-NUMBER IN INVENTORY-MASTER .

e. Directing Computer Arithmetic Function

Using the items illustrated in Figures II-1 and II-2, we shall now direct the computer to change the affected fields in INVENTORY-MASTER to reflect the conditions existing after processing DETAIL.

Assume the records are in memory, disregarding how, and have been matched on STOCK-NUMBER. These are steps necessary to update the item add receipts and subtract shipments from quantity on hand and then multiply the new balance by unit cost to get the new total dollar value of this inventory item. Also we must check the new balance against a predetermined reorder level.

The following is the specific COBOL statements used to accomplish the above steps--please note the similarity.

To add receipts to quantity on hand use the COBOL verb ADD:

```
ADD RECEIPTS TO QUANTITY-ON-HAND.
```

This will change the QUANTITY-ON-HAND in the INVENTORY-MASTER. To subtract the shipments from the quantity on hand use the COBOL verb SUBTRACT:

```
SUBTRACT SHIPMENTS FROM QUANTITY-ON-HAND.
```

This will again change the QUANTITY-ON-HAND in the INVENTORY-MASTER to reflect the new balance.

Now having the new balance the total dollar value must be recalculated using the COBOL verb MULTIPLY:

```
MULTIPLY QUANTITY-ON-HAND BY UNIT-COST
GIVING TOTAL-DOLLAR-VALUE.
```

This will change the TOTAL-DOLLAR-VALUE field of the INVENTORY-MASTER. The INVENTORY-MASTER for stock item A 3456 would now appear as follows:

Stock Number	Description	Quantity on Hand	Unit Cost	Total $ Value	Reorder
A 3456	Fishing Rod, Deep Sea	0075	09.50	00071250	0050

FIGURE II-3

f. Directing Computer Decision Making Functions

The COBOL verb to accomplish decision making is called a conditional. The conditional IF, though not a true verb, functions as a verb.

To continue the problem, check the new balance against the predetermined reorder level and assuming that a routine exists elsewhere in memory to initiate a stock replenishment order. Only if the balance is equal to, or less than, the reorder level must this routine be used. This would be done as follows:

```
IF QUANTITY-ON-HAND IS EQUAL TO REORDER-LEVEL
    GO TO STOCK-REPLENISHMENT.

IF QUANTITY-ON-HAND IS LESS THAN REORDER-LEVEL
    GO TO STOCK-REPLENISHMENT.
```

The result of a condition test must either be met or not met. If the condition is met, the GO TO statement will be executed, if it is not met, a "fall through" to the next statement is executed. In the problem, the QUANTITY-ON-HAND in Figure 3 is not equal to nor less than the REORDER-LEVEL, therefore, the computer would proceed to the next statement--not shown.

As with all programming languages, there are often many ways to do the same thing. This is true of COBOL. For example, in testing for the reorder level above this same job could have been done as follows:

```
IF REORDER-LEVEL IS NOT LESS THAN QUANTITY-ON-
    HAND  GO TO STOCK-REPLENISHMENT.
```

Detail rules governing the use of conditionals will be covered in Section VII and these rules must be observed to achieve maximum use from this flexible and powerful ability of COBOL.

g. Moving Data in COBOL

Data processing involves considerable data movement. The most important kind of data movement is input and output from and to the magnetic tapes, punched cards, or other media on which data is externally stored. The verb OPEN, as in the sentence OPEN INPUT INVENTORY-FILE, establishes communication between the computer and the external file. When the computer has no further need of a file, the verb CLOSE is used to end this communication. These verbs need not be mentioned here, but discussion of them leads to the two verbs used to obtain and dispose of data after communication has been established with the file, the verbs READ and WRITE.

Initially it was assumed that the data was already stored in memory. However, to get it there would require writing of a sequence of steps as follows:

```
OPEN INPUT DETAIL-FILE, MASTER-FILE.
READ DETAIL-FILE RECORD.
READ MASTER-FILE RECORD.
```

When the data is located in storage, the computer can perform the operations we have already specified. Upon completion, the results must be written out in some form. As indicated previously, INVENTORY-MASTER record will contain the updated information after processing has been completed. It may be necessary to move this record to another location from which it can be written upon some output, such as a magnetic tape, a printed form, or a punched card. To make this transfer, we use the verb MOVE as follows:

```
MOVE INVENTORY-MASTER TO UPDATED-
     INVENTORY-MASTER.
```

There must be a properly defined record area called UPDATED-INVENTORY-MASTER and the entire INVENTORY-MASTER will be moved. The data when moved will replace any data previously stored there. The data may be written out by the sentence

WRITE UPDATED-INVENTORY-MASTER.

When an item of data is "moved" a copy of it is made in a new location. The source data remains intact where it was and can be used again. Thus, the computer copied one record into another area in memory. The record as it originally appeared remained unaltered and could be used again.

When an item of data is moved into a location, it will destroy any data stored there. If it is desired to save it, it must be moved out before new data is moved in. This is the reason for the statement MOVE INVENTORY-MASTER TO UPDATED-INVENTORY-MASTER. The record could be written out directly, but occasionally it may be better to move it to an intermediate area first, for reasons not discussed here. A new record is created which may be used later as a replacement for the old INVENTORY-MASTER. The original INVENTORY-MASTER remains intact on tape and can be retained as long as needed.

h. Directing the Computer in COBOL to Perform Cyclical Operations

In the inventory problem, there will be as many INVENTORY-MASTER and as many DETAIL records to process as there are items in stock. The usual case is to process all related records at one time before going on to another phase of the program. In this case, assume it is desired to repeat the processing cycle for as long as there are records to process. It should be noted that names can be given to procedures as well as to data. These names extend to the left of the procedure statements and have significance which will be defined later. It is adequate to note that when the program is printed, this permits the reader to spot the procedure-NAMES at a glance.

In the example which follows, the computer is required to compare two values: STOCK-NUMBER OF INVENTORY-MASTER and STOCK-NUMBER OF DETAIL. Refer back to Figures 1 and

2, note that both the master and the detail records contain items called STOCK-NUMBER. Since identical names were used in the data description, there must be some means of distinguishing between them. The naming system used in COBOL allows this distinction by reference to the name of some larger group of data of which the item is part. The use of an additional name related by the word OF or IN is called qualification. Qualification is required in making distinctions between identical data-names in different records.

An example of repetitive processing cycles and the means to direct their repetition follows:

```
NEXT-DETAIL-ROUTINE.
    READ DETAIL-FILE RECORD.
NEXT-MASTER-ROUTINE.
    READ MASTER-FILE RECORD.
    IF STOCK-NUMBER OF INVENTORY-MASTER IS EQUAL
    TO STOCK-NUMBER OF DETAIL GO TO CALCULATE-
    ROUTINE.
    IF STOCK-NUMBER OF INVENTORY-MASTER IS LESS
    THAN STOCK-NUMBER OF DETAIL GO TO WRITE-
    MASTER-ONLY-ROUTINE.
CALCULATE-ROUTINE.
    ADD RECEIPTS TO QUANTITY-ON-HAND.
    SUBTRACT SHIPMENTS FROM QUANTITY-ON-HAND.
    MULTIPLY QUANTITY-ON-HAND BY UNIT-COST
    GIVING TOTAL-DOLLAR-VALUE.
WRITE-UPDATED-MASTER-ROUTINE.
    MOVE INVENTORY-MASTER TO UPDATED-INVENTORY-
    MASTER.
    WRITE UPDATED-INVENTORY-MASTER.
    GO TO NEXT-DETAIL-ROUTINE.
WRITE-MASTER-ONLY-ROUTINE.
    MOVE INVENTORY-MASTER TO UPDATED-INVENTORY-
    MASTER.
    WRITE UPDATED-INVENTORY-MASTER.
    GO TO NEXT-MASTER-ROUTINE.
```

FIGURE II-4

In this sequence, some of the statements previously used have been omitted.

In Figure 4, each detail record is matched with its corresponding master record. This match is determined by comparing the two STOCK-NUMBERs. The description could have been compared, but this would take longer, as more characters would be compared. The words IF STOCK-NUMBER OF INVENTORY-MASTER IS EQUAL TO STOCK-NUMBER OF DETAIL will cause the computer to compare these values. If they are not equal, the computer will proceed to the next sentence, which causes a test to determine if STOCK-NUMBER OF INVENTORY-MASTER IS LESS THAN STOCK-NUMBER OF DETAIL.

In this example, it is assumed that both the master and detail records are arranged in ascending order, and not allowing for the possibility that any record is out of sequence. A means of continuing to the next part of the program, or of stopping the computer when the last record has been processed, is not provided. This would have to be done in an actual program.

If the stock numbers are not equal, the records do not correspond and processing should not be done. The next step is to get the next master record and determine whether it corresponds to the detail record. As specified, the computer should transfer to NEXT-MASTER-ROUTINE if STOCK-NUMBER OF INVENTORY-MASTER is found to be LESS THAN STOCK-NUMBER OF DETAIL. Upon this transfer, the computer will read the next INVENTORY-MASTER and then repeat the same tests over again. If the new INVENTORY-MASTER does not match the DETAIL, the computer will continue reading in new masters and testing until a match is found.

When the records match, the computer will make necessary changes to INVENTORY-MASTER, move the revised INVENTORY-MASTER to UPDATED-INVENTORY-MASTER, write out this record, and transfer to NEXT-DETAIL ROUTINE. This cycle will continue indefinitely, and additional instructions are necessary to stop it when the last record in the file has been updated.

The ability to repeat a series of instructions increases the power of the computer. It has been shown tests and comparisons can be used in making the computer repeat a procedure. The principles are based on those already considered. There are also several other possibilities for controlling a repetitive procedure.

The verb PERFORM is similar to the verb GO in many ways. Like GO, PERFORM causes a transfer to the first sentence of a routine. Unlike GO, however, the PERFORM verb causes the computer to transfer back, after it has completed the specified procedure, to the next statement following the PERFORM statement. Also, it provides several methods for determining the number of times the procedure is to be performed.

A PERFORM statement can specify a single paragraph to be performed, or, if the procedure consists of more than one paragraph, it can specify two names to identify the beginning and the end of the procedures. To illustrate the usage, of PERFORM, a rewrite of Figure 4 follows:

```
NEXT-DETAIL-ROUTINE.
    READ DETAIL-FILE RECORD.
NEXT-MASTER-ROUTINE.
    READ MASTER-FILE RECORD.
    IF STOCK-NUMBER OF INVENTORY-MASTER IS EQUAL
    TO STOCK-NUMBER OF DETAIL GO TO CALCULATE-
    ROUTINE.
    IF STOCK-NUMBER OF INVENTORY-MASTER IS LESS
    THAN STOCK-NUMBER OF DETAIL PERFORM
    WRITE-UPDATED-MASTER-ROUTINE.
    GO TO NEXT-MASTER-ROUTINE.

CALCULATE-ROUTINE.
    ADD RECEIPTS TO QUANTITY-ON-HAND.
    SUBTRACT SHIPMENTS FROM QUANTITY-ON-HAND.
    MULTIPLY QUANTITY-ON-HAND BY UNIT-COST
    GIVING TOTAL-DOLLAR-VALUE.
WRITE-UPDATED-MASTER-ROUTINE.
    MOVE INVENTORY-MASTER TO UPDATED-INVENTORY-
    MASTER.
    WRITE UPDATED-INVENTORY-MASTER.
GET-DETAIL-ROUTINE.
    GO TO NEXT-DETAIL-ROUTINE.
```

FIGURE II-5

After reading both files, the computer will test to determine if the STOCK-NUMBERs are equal. If they are, control is transferred to the CALCULATE-ROUTINE and continues through the GO TO NEXT-DETAIL statement. If the STOCK-NUMBERs are not equal control "falls through" to the next test. When it is determined that the INVENTORY-MASTER is low, the computer is directed to PERFORM the WRITE-UPDATED-MASTER-ROUTINE. This involves a transfer of control to the first statement and continues to the last statement. This means that the computer does not execute the GET-DETAIL-ROUTINE but instead returns to the next statement after the PERFORM in the NEXT-MASTER-ROUTINE.

The PERFORM verb allows for many ways of performing a procedure. For example, the statement containing the PERFORM verb can stipulate the number of times the procedure is to be performed, or it can stipulate that the procedure is to be performed until a specified condition exists. If no such stipulation is made, the computer will interpret the command as if the number of times had been stipulated as 1.

To illustrate, refer to Figure II-5. Instead of specifying the actual test to determine whether the records match, and then stating the required action, it could be written as follows:

```
IF STOCK-NUMBER OF INVENTORY-MASTER IS LESS
    THAN STOCK-NUMBER OF DETAIL PERFORM
    NEXT-MASTER-RECORD ROUTINE UNTIL STOCK-
    NUMBER OF INVENTORY-MASTER IS EQUAL TO
    STOCK-NUMBER OF DETAIL.
```

This statement could be simplified still further, thusly:

```
PERFORM NEXT-MASTER-RECORD-ROUTINE UNTIL
    STOCK-NUMBER OF INVENTORY-MASTER IS
    EQUAL TO STOCK-NUMBER OF DETAIL.
```

The phrase beginning with the word UNTIL stipulates the condition which must be met to end the performance of this procedure. It should be noted in the discussion of the PERFORM verb that this is just one of many ways of accounting for the number of times a procedure is performed. Each has its own special feature.

3. RELATING THE PROGRAM TO THE UNIVAC 1107

In general, the structure of the COBOL language resembles English and business problems may employ it directly. COBOL is "problem oriented" (i.e., created to satisfy the expressions used in solving business problems) not "machine oriented" (i.e., organized to satisfy the "bit" understanding of a computer).

A processor is used "to translate" a COBOL language program into a computer language program.

In order to use the UNIVAC 1107 effectively, a programmer must know the internal characteristics of the UNIVAC 1107 central processor; i.e., its word length, internal codes, etc. This knowledge is particularly helpful in writing the data descriptions.

The portion of COBOL most affected is the Environment Division. It is here that the problem is related to the UNIVAC 1107 system. A programmer describes the particular UNIVAC 1107 configuration the problem is to use. (See Section VIII, Environment Division, for details.)

4. CREATING THE OBJECT PROGRAM

It has been pointed out that a COBOL language program will have to be translated into a machine language program before it can be used to process data. This translation is accomplished as follows:

A UNIVAC 1107 COBOL processor is provided. A processor is actually a special program which analyzes the words and characters of a COBOL language program and creates a new program in the internal language of the UNIVAC 1107. A single COBOL language statement--sometimes even a single word or character--will often produce a great many machine instructions. It is the function of the processor to determine what these instructions should be and to combine them to form a new program. The processor must also take care of such supporting details as reserving space in storage for data and instructions and providing a means of identifying each item.

The processor is stored in the computer first. Then the source program (i.e., the COBOL language program) is read in and the processor analyzes it and creates the object program--the machine language program. The object program is prepared in a form suitable for the UNIVAC 1107, recorded on magnetic tape. When the object program has been prepared, it may be read back into the computer at once, or it may be stored externally for future use. Before data can be processed, the object program must be read back into the computer's internal storage. The system is then ready to process data.

Since the object program is a machine language program, the processor must take into account the operating characteristics of the machine and the capacity of the equipment available to it. Each installation will differ in some way from others of its type. For example, the storage capacity available for storing data and instructions internally may be greater in one installation than in another. Furthermore, the number of tape units available for external storage of data may vary, either as to the number of units actually installed or the number available for the particular job.

It is therefore necessary to furnish the processor with certain basic information about the equipment available to it. This is done by writing the proper COBOL language statements in a portion of the program called the Environment Division. The rules for doing so are set forth later in this manual.

The Environment Division also provides the programmer with the capacity to assign names to various units of the equipment so that these names can be used in the procedure statements. For example, if the programmer wishes certain information to be written out on a typewriter, he can write such a procedure statement as DISPLAY RESULT UPON TYPEWRITER, provided he has identified the typewriter by name in the environment description.

5. DIVISIONS OF THE COBOL SOURCE PROGRAM

a. General

A COBOL source program (program which has not been compiled) is composed of four elements:

1) The identification of the program.

2) The description of the equipment being used in the processing.

3) The description of the data being processed.

4) The set of procedures which determine how the data are to be processed.

The COBOL system has a separate division within the source program for each of these elements. The names of these divisions are: IDENTIFICATION, ENVIRONMENT, DATA, PROCEDURE. A brief description of each follows.

b. Identification Division

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

c. Environment Division

The ENVIRONMENT DIVISION is that part of the source program which specifies the equipment being used. It contains a description of the computer to be used both for compiling the source program and for running the object program. Memory size, number of tape units, printers, etc., are among many items that may be mentioned for the UNIVAC 1107. Problem oriented names may be assigned to a particular equipment. Those aspects of a file which relate directly to hardware are described here. Because this division deals entirely with the specifications of the equipment being used, it is largely computer dependent.

d. Data Division

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

e. Procedure Division

The PROCEDURE DIVISION specifies the steps that the user wishes the computer to follow. These steps are expressed in terms of meaningful English words, statements, sentences, and paragraphs. This aspect of the overall system is often referred to as the "program"; in reality it is only part of the total specification of the problem solution (i.e., the program)

and is insufficient, by itself, to describe the entire problem. This is true because repeated references must be made--either explicitly or implicitly--to information appearing in the other divisions. This division, more than any other, allows the user to express his thoughts in meaningful English. Concepts of verbs to denote actions, and sentences to describe procedures, are basic, as is the use of conditional statements to provide alternative paths of action. The PROCEDURE DIVISION is essentially computer-independent.

6. NOTATIONS USED IN VERB AND ENTRY FORMATS IN THIS MANUAL

Throughout this manual, basic formats are given for the various verbs, clauses, entries and other essential elements of the COBOL language. These formats must be followed when writing a COBOL program.

Rules of notation are as follows:

a. All uppercase words which are underlined are required when the functions of which they are a part are used. An error will be indicated at compilation time if the underlined words are absent or incorrectly spelled.

b. All upper case words which are not underlined are optional but belong to the COBOL language and are used for readability only.

c. All lower case words represent generic terms which must be supplied by the user, when the clauses of which they are a part are used.

d. Material in braces { } indicates that a choice from the contents must be made.

e. Material in square brackets [ ] represents an option and may be included or omitted at the user's choice.

f. Notes will elaborate on the formats and specify any restrictions.

g. In cases where many choices are available, some separations into numbered options have been made.

h. When two or more nouns may be written in a series, commas (,) are shown as connectives. Where a comma is shown in the formats, it may be omitted or replaced by either "AND" or " , AND".

i. Punctuation, where shown, is essential. Other punctuation may be inserted in accordance with rules specified in the procedure.

III. SUGGESTED COBOL DOCUMENTATION

1. GENERAL

Documentation of computer applications has normally been additional effort after the application has been installed and operating. This fact has always been detrimental to effective, adequate documentation. The methods of documentation that follow have been designed to be an aid to, or a by-product of the design and programming of the application rather than added work after an application is installed.

2. APPLICATION DOCUMENTATION

a. General

The documents which pertain to the entire application are called Application Documentation. They include the formats of the input and output records or reports, definitions of, or references to coding systems used in the data and a chart of the individual programs showing the flow of data. These elements of the application documentation are normally produced in parallel as each is an aid to the production of the others.

b. Application Chart

1) General

An application chart is a graphic portrayal of the programs that constitute an application and which illustrates the flow of data and its timing. The application chart can be used as a basis for estimating the required programming effort, computer time and scheduling.

The first chart drawn for an application may be revised many times as a broad analysis of each program is made in conjunction with the development of the report and record layouts. As programming progresses and detailed analysis of each program is made possible, further revisions may

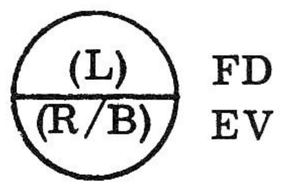
be necessary. Even though the application chart is subject to many changes, a tentative chart should exist before any programming is initiated.

The application chart is drawn on a vertical scheme using the charting symbols and notations described below.

2) Files

The following symbols and notations are used to designate the various input-output files when drawing a computer application chart.

a) Tape File

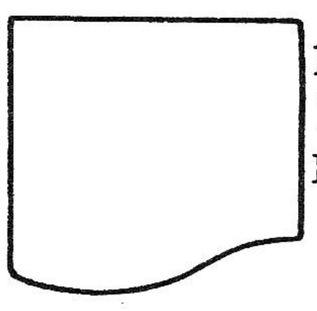


L is the value of identification (the label)
FD is the file description name
R/B is the record size and blocking factor
EV is the estimated volume in records

b) Card File

FD is the file description name
EV is the estimated volume

c) Printer Output



FD is the file description name
I is the information on description and distribution
EV is the estimated volume in lines

d) Paper Tape File

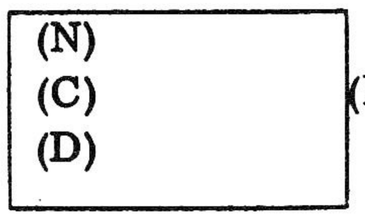
FD is the file description name

EV is the estimated volume in records

3) Hardware

The following symbols and notations are used to designate the various computer components (hardware) when drawing a computer application chart.

a) Computer



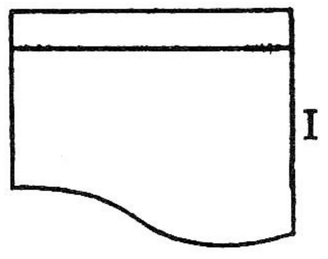
N is the program number

C is the program classification (i.e., Sort, Edit, Maintenance, etc.)

D is the program description

KEY is the name of the control fields for sort and merges

b) Display Output



I is the description of the information displayed and distribution information

4) Sample Application Chart

The operation cycle (daily, weekly, monthly, etc.) is shown in the title of each portion of the application chart which differs in this regard.

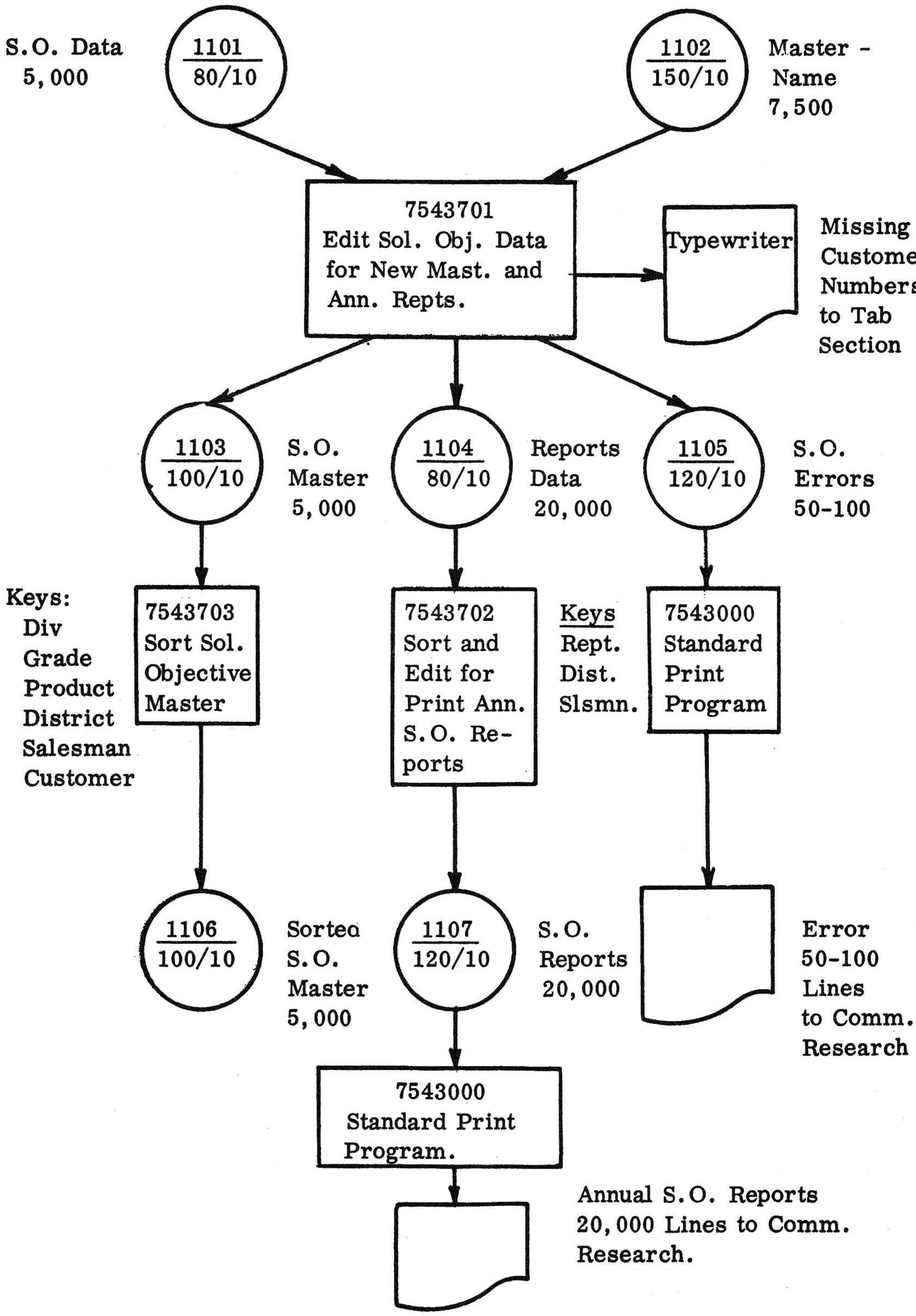
Copies of the chart are distributed and maintained for:

a) The organization which maintains the program

b) The customer

c) The Data Processing Center

Outlined below is a sample application chart.



c. Report Layouts

For each report produced by an application the documentation shall include:

1) Form layout showing spacing and printing positions, headings and any variations they may have, and other than normal total positions,

2) Control information such as schedules, consumption, distribution, and retention which must be handled by the computer operating department,

3) Forms specifications such as form number, size, and number of copies,

4) Sample of the report.

It is suggested that the above documentation be duplicated in the Operator's Instructions for the program which produces the report.

d. Record Layouts

All external files (tape input or output, cards in or out, etc.), working storage areas for records, and constant records are graphically recorded to provide documentation which fulfills COBOL Data Division Requirements. The method required is the horizontal display of the record as illustrated with the sample program. Each record is documented once for the application and must exhibit:

1) An English File Name. (The name of the file in which the record is found).

2) An English Record Name.

3) A list of programs in which the record is used or produced.

4) English names for each item and group of items within the record according to COBOL Data Division requirements. Data names must also be as meaningful as possible. Where a lower level language is used to produce the machine language, the tag assigned to conform to the files of the lower level language is also given, e.g. tags for SLEUTH II.

5) Each item must have the size, class, and other requirements given. Usage of the picture clause provides these details.

6) Special record codes, etc., are exhibited where room permits, otherwise a reference to a coding document is supplied when unique to this record.

7) Record layout of data arrangement within words.

3. RUN DOCUMENTATION

a. General

The documents which pertain to one program of an application are called Run Documentation. Those documents include general and detail charts of the program logic, listings of the program, data for testing the program, instructions for operation, and a history of changes to the program.

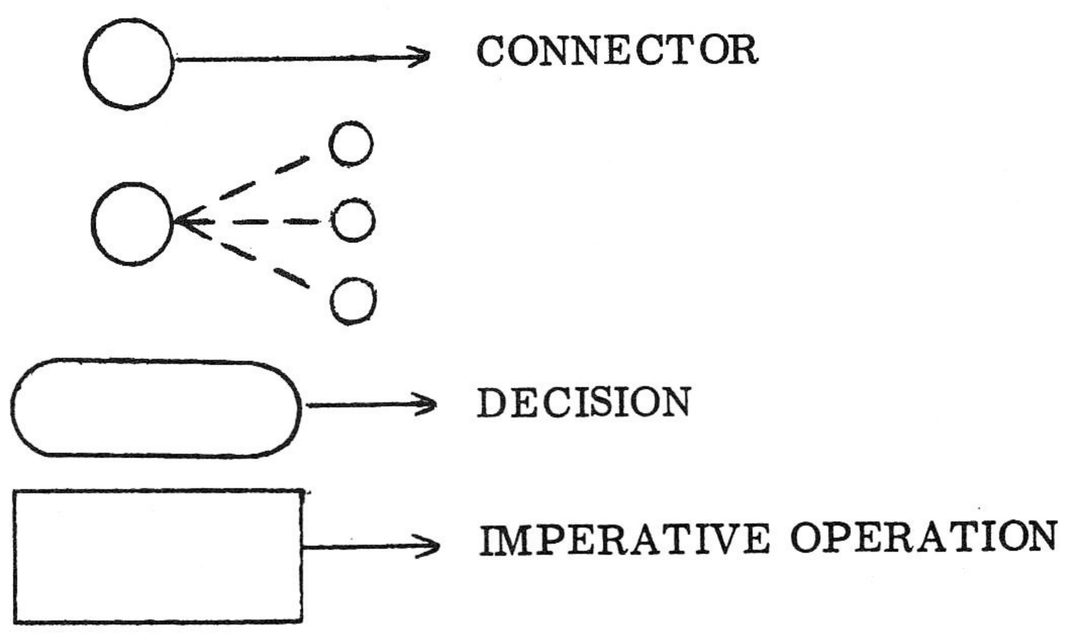
b. Run Chart

The run chart is a copy of that portion of the application chart which shows the inputs and outputs of the particular program. The run chart is drawn on the Form "Flow Chart of Input and Output Data", which is included in the sample program. In addition to the actual flow chart, other pertinent data such as program number, programmer, type of run, core position used, run time, frequency run, program name, program purpose and tape identification are shown on the run chart.

c. Block Chart

A block chart, the general organization of the logic of a program, is usually originated in one of two ways. The first is the product of a general problem description developed for a programmer by someone who desires to communicate with the programmer in chart form rather than in narrative form. The second is the result of preplanning on the part of the programmer who uses it to organize his solution and check the problem definition. In either case the amount of detail varies almost directly with the individual and the problem. Utilization of this tool is required as a work habit. The following rules are helpful:

1) The following symbols and notations are to be used:



2) Narrative English language is used.

3) A horizontal scheme is required.

4) Vertical branches must be restricted to short loops, connectors, etc.

5) A maximum of four lines may be diagrammed on a single 11 x 17 page.

6) Each horizontal line must begin and end with a connector.

7) Each connector shall contain a number representing the page number as the left side, and a connector number as the right side; i.e., 1.1 represents page one, connector 1. These connectors at the end of a

line or representing a branch are given the number of the sequential connector that control passes to when this path is followed.

8) Each page of the block chart should contain the program number, page number and data or date reused.

9) Fail sides of all logical statements generally must run from left to right.

10) Block charts should be brief.

The only exception to the above rules exists for programs produced from a report generator, sort or merge package. The programs require charts only for the additional data which has been added to the original program.

An example of the English Language Block Chart is included with the sample program.

d. Record and File Descriptions

1) Record Description

Each input and output record should be recorded on the COBOL Record Form which is exhibited with the sample program. In addition to the detail layout of each record, the following information is shown on the form:

a) File Name
b) Format Code
c) Issue Date
d) Revision Date
e) Label Code

2) File Description

A form titled "File Description by Program" should be prepared for each COBOL program written. The following data is shown on the form:

a) Program Name
b) Date
c) Program Code
d) File Description Name
e) Label Code
f) Format Code
g) Record Source
h) Any pertinent comments or remarks

An example of this form is included in the "Sample Problem portion of this manual.

e. COBOL Chart

1) General

The COBOL Chart is a detail logic chart of a program. All tape computer programs other than programs produced from Sort, Merge, Report or other program generators require a COBOL chart. If a program produced from a generator has any additional logic, the addition must be covered by a COBOL Chart.

In addition to the general rules which apply to the Block Chart, the COBOL Chart requires observance of the following rules:

a) Every change to the computer object program should be reflected in the COBOL Chart.

b) Data names used must meet the requirements of COBOL and every effort must be made to make them meaningful.

c) A copy of the COBOL Chart should be kept at the archives.

An example of the COBOL Chart is included in the sample problem.

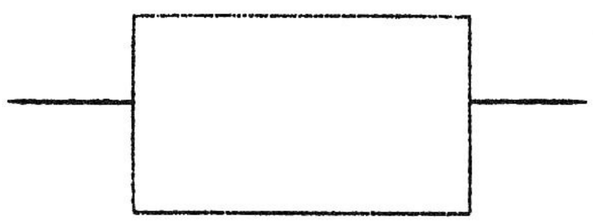
2) Symbols

a) General

COBOL Charts are made using COBOL verbs and symbols for the verbs as outlined below. A detailed explanation of the COBOL verbs is found in Section VIII Procedure Division. A complete understanding of the verb meanings must be acquired prior to the preparation of a COBOL Chart. This is the basis for uniform problem definition.

b) Symbol For Imperative Operations

The COBOL symbol for use with imperative operations is as follows:



This symbol is used with the following COBOL verbs:

ACCEPT	EXAMINE	STOP
ADD	EXIT	SUBTRACT
ALTER	GO (Unconditional)	WRITE
CLOSE	MOVE	
DISPLAY	MULTIPLY	
DIVIDE	NOTE	

The full and exact format of the verb statement will be written within the symbol.

c) Symbol For Conditional or Decisive Operations

The COBOL symbol for use with Conditional or Decisive operations is as follows:

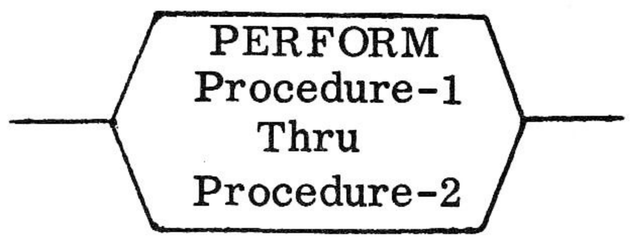
This symbol is used with the following COBOL verbs:

READ
IF
GO (Conditional)

A separate symbol must be used for each IF clause even though they may be in the same statement.

d) Symbol For The PERFORM Verb

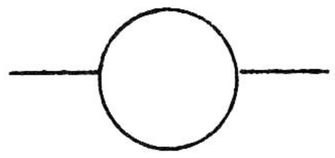
The PERFORM verb has a unique symbol which must not be used for any other verb. The symbol is as follows:



e) Symbols For Connectors

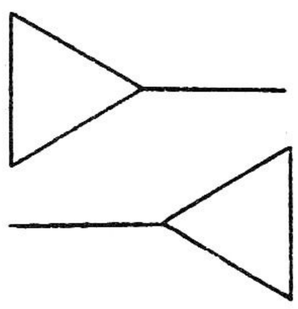
COBOL connectors use three symbols. Connector must be used for every transfer or entrance of control but it is recommended that a connector be used with each COBOL symbol.

(1) The connector for section names or paragraph names is as follows:



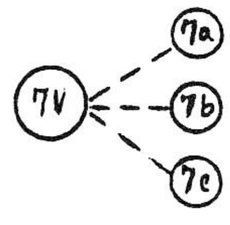
COBOL paragraph names and section names are written above the connector when required.

(2) The connector for entrance or exit to closed sub-routines is as follows:



The entrance only carries the connector number. These sub-routines are entered by using the PERFORM verb.

(3) The connector to be used to denote switches is as follows:



COBOL charts may contain cross-referencing by noting the connector number which transfers control below the connector which is the entry of the control transfer.

f. Program Listings

The coding system used produces certain listings to document the program. These listings are retained and replaced periodically according to local conditions and practices. Where COBOL compilers are in use, the COBOL Reference Format either used as input or as produced by the Compiler should be retained in some manner, e.g., cards, tape or printed listings. Corrections or maintenance changes are reflected on these listings as well as on the COBOL Chart.

g. Test Data

The data created to test all paths of the program and/or application logic must be retained either as part of each program's documentation, or as a portion of the application documentation with cross references to programs for which the data was created. This information must be maintained as the programs are maintained, and be available as a checkpoint for auditing of the system.

h. Operator's Instructions

Operator Instructions for all computer operations as well as for the operation of all auxiliary equipment should be prepared.

Certain program operating instructions shall be provided along with a COBOL compiler which encompass such things as tape error routines, label check procedures, etc. In lieu of a COBOL compiler, certain standard operating instructions shall be a by-produce of the conventions adopted. Where possible, standard operator instructions should apply to all programs and all applications for the same type of computer system. Any unique requirements of the computer operator are listed with the instructions for the program to which they apply. In addition to any unique instructions, significant memory addresses are usually of the greatest aid to the operator. As a general policy this information should be kept to a minimum. Operator requirements to complete programs are highly undesirable, and every effort must be taken to program around the operator to solve all contingencies. Information on timing retention, distribution, special conventions, labeling, re-run procedures, and variables used in the solution of a program are useful to the operator. Instructions for the preparation of special control data and for recording of program times should also be provided when desirable and/or required.

In addition to the above, the operator should be provided with a duplicate of the application chart.

i. Historical Record

Each program shall have filed with it a historical log of the changes made to the program. This log shall exhibit:

1) Date of Change (first used in altered condition).

2) Programmer's Name.

3) Description of Change (including description of errors corrected, etc.).

4) How Need for Change was Established. (Who initiated the request?)

5) Who Authorized the Change? (Absence of information shall imply the programmer made the decision.)

IV. COBOL LANGUAGE STRUCTURE AND USAGE

1. GENERAL

COBOL is built up from its smallest possible units, a set of characters (letters, numbers, punctuation marks, etc.) just as the English language. These characters are used to form meaningful words by following certain rules. In English different types of words are found such as nouns, verbs, adjectives, etc. The types are combined to form expressions, statements, sentences, and paragraphs. The COBOL language also consists of different types of words, and they are combined, using COBOL's rules of syntax, into clauses, sentences, etc.

This starts with the smallest part of COBOL, the character, and then builds words, expressions and the larger units of the language. The rules presented here must be thoroughly understood and observed before COBOL programs can be constructed. To produce a program in COBOL, the programmer must use the exact language and rules correctly and precisely.

2. CHARACTER SET

a. Complete Character Set

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

0,1, ... , 9
A,B, ... , Z
Blank or space
\+ Plus sign
\- Minus sign or hyphen
* Asterisk
/ Stroke (virgule or slash)
= Equal sign
$ Dollar sign
, Comma
. Period or decimal point
; Semicolon ---- substitute character : colon
" Quotation mark ---- substitute character ' apostrophe
(Left parenthesis
) Right parenthesis
> "Greater than" symbol
< "Less than" symbol

The colon has been designated as an alternate character for the semicolon, and the apostrophe as an alternate character for the quotation mark.

Throughout the context of this manual, the words "semicolon" and "quotation mark" will be used to refer either to these characters or their substitutes.

b. Characters Used For Words

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

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

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

c. Characters Used For Punctuation

The punctuation characters consist of the following:

" Quotation mark
(Left parenthesis
) Right parenthesis
Space or blank
. Period
, Comma
; Semicolon

d. Additional Characters Used In Editing

$ Dollar Sign
* Check Protection Symbol
, Comma
. Actual Decimal Point

e. Summary of Allowable Characters

Those characters which are recognizable by the COBOL system include the letters of the alphabet, decimal integers, and those characters which are used in editing.

3. WORDS

a. Definition of Words

A word is a combination of characters chosen from the following set:

0 through 9
A through Z
- (hyphen; a word can neither begin nor end with a hyphen).

A word may have any number of characters up to 30. The last character of a word must be followed by a space, a period, a right parenthesis, a comma, or a semicolon. When the last character of a word is followed by any of these other than a space, the punctuation character must be followed by a space.

When a punctuation character is used with a word, there must be no space between that character and the word. That is, the left parenthesis and the beginning quotation mark,when used, must be followed immediately by the word with no intervening space; the period, comma, semicolon, right parenthesis and the ending quotation mark must follow the word immediately, with no space intervening.

There is a single exception to this rule; where spaces are desired in a literal, these spaces will be enclosed with the word within the quotation marks. (See "Literals", this section)

b. Types of Words

1) Nouns

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

Data Name
Condition Name
Procedure Name
Literal
Figurative Constant
Special Register Name
Special Name

Since a space signifies the end of a word in UNIVAC 1107 COBOL word analysis, two or more words may be connected with a hyphen or hyphens to create a single word.

Example:

stock-number
quantity-on-hand

a) Data-Names

A data-name is a word with at least one alphabetic character, which designates any data specified in the data description and by means of which data are referenced in procedure statements.

b) Condition-Names

If a data-name may have many different values, it is called a conditional variable. A condition-name is a name assigned to a particular value, set of values, or range of values which a data-name (i.e., the conditional variable) may have at a given time. A condition-name must contain at least one alphabetic character.

A condition-name must be unique, or be made unique through qualification; the conditional variable may be used as the qualifier. If references to a conditional variable require subscripting, then references to any of its condition-names also require subscripting. (See "Qualification" and "Subscripting" in this section, for details.)

The VALUE clause is the only clause which may be used in a condition-name data description. Condition-names may be used only in conditional statements.

Example:

```
02 TITLE

    88  ANALYST VALUE IS 1.
    88  PROGRAMMER VALUE IS 2.
    88 CODER VALUE IS 3.
```

The conditional statement:

IF CODER.....

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

c) Procedure-Names

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

d) Literals

A literal is a data item whose actual value is identical to the characters comprising it. A literal may have a maximum length of 132 characters. All CLASS rules for data-name apply to literals. A literal may belong to one of two classes: non-numeric, (i.e., alphabetic or alphanumeric), or numeric. Non-numeric literals must be bounded by quotation marks. Any literal bounded by quotation marks, even though it conforms to the rules for numeric literals, will be considered a non-numeric literal. That is:

1234 is not the same as "1234"

A non-numeric literal is composed of any allowable characters except the quotation mark. All spaces which are enclosed in the quotation marks are included as spaces in the literal.

A numeric literal is composed only of characters chosen from the numerals 0 through 9, the plus (+) or minus (-) sign, and the decimal point. The rules for formation of numeric literals are:

(1) A numeric literal must contain only one sign character and/or one decimal point.

(2) The literal must contain at least one digit.

(3) The sign in the literal must appear as the left-most character of the literal. If the literal is unsigned, the literal is considered to be positive.

(4) The decimal point may appear anywhere within the literal except as the right-most character of the literal, and is treated as an implied decimal point. If the literal contains no decimal point, the literal is considered to be an integer.

Examples:

Use of a non-numeric literal

```
STOP "INPUT SHOULD BE MASTER".
```

Use of a numeric literal

```
ADD 1234 TO TONS.
```

Note also that any characters (even those which have been represented as a reserved word or verb) may appear as a non-numeric literal. (See first example of non-numeric literals; i. e., EXAMINE is also a verb).

e) Figurative Constants

For ease in programming, certain values, which are often used as "fill", have been assigned fixed data-names. These values are called "figurative constants".

The singular and plural forms of figurative constants are equivalent, and may be used interchangeably. The fixed data-names and their meanings are as follows:

ZERO ZEROS ZEROES	Represent the value 0.
SPACE SPACES	Represent one or more blanks or spaces.
QUOTE QUOTES	Represent a sequence of quotation mark characters.
ALL "any literal"	Represents a sequence of "any literal".

Examples:

Assume that PART-NO is a six character field.

```
MOVE ZERO TO PART-NO.
     PART-NO will be 000000
MOVE SPACES TO PART-NO.
     PART-NO will be blank
MOVE QUOTE TO PART-NO.
     PART-NO will be """"""
```

The word QUOTE cannot be used instead of the quotation mark to bound a literal.

QUOTE NAME QUOTE instead of "NAME" is incorrect. If we say:

```
DISPLAY "NAME".
```

The result will be:

```
NAME
```

If we want NAME to appear in quotation marks, we can achieve this by treating QUOTE and NAME as a series of literals, in a DISPLAY statement.

DISPLAY QUOTE, "NAME", QUOTE.

The result will be

"NAME"

Example of use of ALL "any literal":

MOVE ALL "4" TO PART-NO.

PART-NO will be 444444

The names, when used as figurative constants, must not be bounded by quotation marks. If the names are bounded by quotation marks, they will be considered as non-numeric literals.

Thus:

MOVE ALL "ZERO" TO PART-NO. will result in ZEROZE in the six character PART-NO field.

The programmer is not limited to filling with zeros, spaces, or quotes; the ALL "any literal" option allows him to fill with <u>any</u> character or series of characters, as represented in "any literal".

f) 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 information produced by the EXAMINE verb. It may also be used, however, to hold information produced elsewhere in a program. In the object program, the TALLY register occupies one full computer word.

Example:

```
01 TALLY SIZE IS 5 COMPUTATIONAL-1
   SYNCHRONIZED RIGHT.
```

The preceding example is that of a Record Description for the TALLY register. For illustration of use of the TALLY register, see Section VII, EXAMINE.

g) Special Names

Special names provide a means of relating hardware with problem-oriented names (See Section VIII and Section X).

2) Verbs

A verb usually denotes action. Whenever a verb is used, some action will take place. However, not all verbs cause action to take place at object time. Some verbs are compiler-directing verbs or processor verbs. For example, sometimes it is helpful to include information which is not used in producing the object program, but which comments on the associated COBOL statements to make a listing of the program more intelligible to the reader. The processor verb NOTE precedes a message which will appear in the listing of the program. This message, however, will have no effect on the function of the object program. The verb NOTE, therefore, does not cause action in the object program.

However, normally, verbs which are not compiler directing verbs will cause action at object time. The verb ADD will cause specified information to be added, and the verb STOP will cause the computer or program to halt.

The word IF, though not a true verb, may cause action provided the condition associated with the IF is met. If the condition is not met, no action is taken. Rules governing conditions are found in the PROCEDURE DIVISION, Section VII.

3) Reserved Words

Reserved words are used for syntactical purposes and may not be used as nouns or verbs. There are three types:

a) Connectives

Connectives are words used to:

(1) Denote the presence of a qualifier: OF, IN (see Qualifiers, following).

(2) Form compound conditions: AND, OR; these are called logical connectives.

b) Optional Words

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

For example, all of the following statements are correct and have identical meaning, while only the last statement does not contain any optional words:

```
IF  X  IS EQUAL TO  Y --
IF  X  EQUAL TO  Y  --
IF  X  IS EQUAL  Y  --
IF  X  EQUAL  Y  --
```

c) Key Words

Key words are words which must be present to convey the meaning of the statement to the compiler. A key word cannot be omitted.

Key words are of three types:

(1) Verbs: ADD, READ, ENTER, etc.

(2) Required words, appearing in formats in various divisions of the language, needed to complete the meaning of certain verbs or entries: TO, OMITTED, MEMORY, etc.

(3) Words not shown in any format, but which have a specific functional meaning: NEGATIVE, SECTION, TALLY, etc.

4) Qualifiers and Subscripts

a) Qualifiers

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

Qualification in COBOL is performed by appending one or more prepositional phrases, using IN or OF. The choice between IN or OF is

based on readability, because they are logically equivalent. Nouns must appear in ascending order of hierarchy with either of the words IN or OF separating them. The qualifiers are considered part of the name. Thus, whenever a data item or procedure paragraph is referenced, any necessary qualifiers must be written as part of the name.

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

```
01 MASTER .....
   02 CURRENT-DATE .....
      03 MONTH .....
      03 DAY .....
      03 YEAR .....
   02 LAST-TRANSACTION-DATE ...
      03 MONTH .....
      03 DAY .....
      03 YEAR .....
```

```
01 NEW-MASTER .....
   02 CURRENT-DATE .....
      03 MONTH .....
      03 DAY .....
      03 YEAR .....
   02 LAST-TRANSACTION-DATE ...
      03 MONTH .....
      03 DAY .....
      03 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 each hierarchy (the 03, 02, and 01) in order to make the elementary item (month and day respectively) unique.

This is analagous 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 Oskosh 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:

(1) A qualifier must be of a higher level and within the same hierarchy as the name it is qualifying. (See "Concept of Levels", Section VI, for further discussion on hierarchies.)

(2) The same name may not appear at two levels in a hierarchy so that it would appear to qualify itself.

(3) If a data-name or condition-name is assigned to more than one data item in a program, it must be qualified in all references to it in the PROCEDURE DIVISION and the ENVIRONMENT DIVISION, and in the DATA DIVISION. (See individual entries in Section VI for details.)

(4) Any data-name requiring qualification must be qualified every time it is referenced, i.e., in the absence of qualification, the COBOL compiler cannot determine the logical reference.

(5) A paragraph-name must not be duplicated within the same section. A paragraph-name can only be qualified by a section-name. When it is, the word SECTION must not appear. A paragraph-name need not be qualified when referred to from within the same SECTION. Subscripts and conditional variables, as well as procedure and data-names, may be made unique by qualification where necessary or desirable.

(6) A data-name cannot be subscripted when it is being used as a qualifier.

(7) A name can be qualified even though it does not need qualification. The use of more names for qualification than are actually required for uniqueness is permitted. If there is more than one combination of qualifiers which insure uniqueness, then any set can be used.

(8) The name of a conditional variable can be used as a qualifier for any of its condition-names.

b) Subscripts

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

A subscript is an integer whose value determines which element is being referred to within a table (or list) of like elements. The subscript may be represented either by a literal which is an integer (e.g. 25), or by a data-name (e.g. AGE) which has an integral value.

When the subscript is a data-name, it must be described by a Record Description entry in the DATA DIVISION. In both cases; i.e., whether the subscript is a literal or a data-name, the subscript is enclosed in parentheses, and appears immeidately after the space ending the name of the element referenced e.g. RATE (AGE) or RATE (25). Tables are often defined so that more than one level of subscripting is required to locate an element within them. A maximum of three levels of subscripting is permitted by COBOL. Multi-level subscripts are always written from left to right in order: Major, intermediate, minor. In this case the subscripts are shown in a single pair of parentheses, and separated by commas. For example:

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

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

NOTE: For particulars on use of subscripts, see 9.5 of this section.

c) Distinction Between Subscripts and Qualifiers

There is a distinct difference between qualification and subscripting. Qualification is necessary when the same data-name is used for several different items of data; subscripting

is necessary when some of the elements of a table or list have not been assigned individual names (see the OCCURS clause in Section VI).

In actually subscripting a data-name, the following rules are significant:

(1) The qualifiers are considered part of the data-name.

(2) A data-name being used as a qualifier cannot be subscripted.

(3) Qualification not logically required for uniqueness is permitted.

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

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

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

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

5) Series Connectives and Logical Connectives

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

a) Logical Connectives

AND

OR

b) Series Connectives

AND

,

, AND

4. STATEMENTS

a. General

The next unit of COBOL language is the "statement". The statement in COBOL is similar to the clause in English. In its simplest form, a statement contains a verb and its operands. The word "operand" is used hereafter to specify the item acted upon by the verb.

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

b. Imperative Statements

Imperative statements direct the computer to perform specific actions. An imperative statement consists of one or more "commands". It is made up of either a verb (excluding compiler directing verbs) and its operands, or a sequence of imperative statements. A sequence of imperative statements may contain either a GO imperative statement or a

STOP RUN imperative statement, which (if present) must appear as the last imperative statement of its (GO or STOP RUN sequence.

Examples:
MOVE A TO B
ADD C TO D
GO TO ZILCH

c. Conditional Statements

Quite often it is necessary to modify an imperative statement so its execution will depend on a conditional expression. When one or more conditional expressions are included in an imperative statement; such a statement than becomes a conditional statement.

A conditional statement is evaluated before action is taken. In general, if the conditions are true, the program carries out the action specified in the statement, but if the condition is false, the program passes control to another statement.

There are three basic forms of conditional statements, but in all, action is performed only if a specified condition is met. The three forms and applicable rules are as follows:

1) IF "conditional expression" statement - 1

If the expression is found to be true, statement-1 will be executed. If the expression is found to be false, statement-1 will be by-passed; the program will instead proceed to the next sentence. A conditional statement in this form need not be the only statement in the sentence; however, it must be the last.

Examples:

IF GROSS-PAY IS LESS THAN EXEMPTION-AMOUNT GO TO NO-TAX ROUTINE.

SUBTRACT EXEMPTION-AMOUNT FROM GROSS-PAY GIVING TAXABLE PAY.

In the previous example, the only reason for departing from the next-in-line sequence of program steps is the detection of a specified condition. If the condition is not detected, the next statement is given control. In this case, the SUBTRACT statement.

2) IF condition $\left\{\begin{matrix}\text{statement-1}\\ \underline{\text{NEXT SENTENCE}}\end{matrix}\right\}$ $\left[\left\{\begin{matrix}\underline{\text{OTHERWISE}}\\ \underline{\text{ELSE}}\end{matrix}\right\}\left\{\begin{matrix}\text{statement-2}\\ \underline{\text{NEXT SENTENCE}}\end{matrix}\right\}\right]$

This represents a conditional statement in its fullest form. If the condition is true, either statement-1 or NEXT SENTENCE, whichever is specified, will be executed. Statement-1 must be a simple or compound imperative statement. It can not be a conditional statement. If NEXT SENTENCE is specified, the program will transfer control to the next sentence in the program and ignore the ELSE or OTHERWISE portion of the IF statement.

If the condition is false, the program will take no action until either the word ELSE or the word OTHERWISE, one of which must be written is found. Then the program will execute statement-2 or NEXT SENTENCE, whichever is present. Statement-2 may be one of the following: (a) A simple imperative statement. (b) A compound imperative statement, or (3) a conditional statement. If NEXT SENTENCE is specified, control will pass to the next sentence in the program.

Examples:

```
IF GROSS-PAY-TO-DATE IS GREATER THAN
4800 NEXT SENTENCE OTHERWISE PERFORM
FICA-ROUTINE.

IF GROSS-PAY-TO-DATE IS GREATER THAN
4800 NEXT SENTENCE OTHERWISE IF GROSS-
PAY IS GREATER THAN ZERO PERFORM
FICA-ROUTINE.
```

In the first example, statement-2 is a simple imperative, and in the second example, a conditional statement.

3) Statement-1 {AT END / ON SIZE ERROR} Statement-3
Statement-2

These are special forms of conditional statement, which are used only with arithmetic verbs and the verb READ. The form for their use is incorporated in the formats for those verbs. It is different from the other condition statements due to the nature of the conditional expressions which must be evaluated. These follow:

a) The SIZE ERROR condition

An arithmetic verb may produce a result larger than the space provided for in storage. When this happens, a "size error" occurs, and it may have disastrous effects; therefore, a test to determine if this size error has occurred must be made. The conditional statement used for this purpose is the ON SIZE ERROR. In the format above, statement-2 represents a statement utilizing one of the arithmetic verbs, followed by the words ON SIZE ERROR and then by statement-3. Statement-3 prescribes the action to be followed if a size error is found and must be an imperative statement. It can not be a conditional statement.

b) The AT END condition

When the records in a file are delivered to the object program by READ statements, a particular action is to be used when the end of the file is detected. The words AT END are used to alter the sequence of a program when the end of file is found. In this form, statement-1 represents a READ statement, the words AT END must be followed by statement-3, which indicates the action to be carried out when the condition is found. Statement-3 must not be a conditional statement.

Example:

READ MASTER RECORD AT END
PERFORM CLOSE-MASTER ROUTINE.

4) Compiler Directing Statements

A compiler directing statement consists of a compiler directing verb and its operand. The compiler directing verbs in COBOL are: ENTER, COPY, NOTE, EXIT, RETURN, USE.

Example: ENTER SPECIAL-METHOD-2-ROUTINE.

5. SENTENCES

a. General

A sentence consists of a sequence of one or more statements, the last of which is terminated by a period. The statements comprising the sentence must be either (a) one Compiler Directing Statement or (b) an Imperative Conditional Statement, syntactically correct according to the rules outlined in Statement 4. There are three types of sentences: Imperative, Conditional and Compiler Directing.

b. Imperative Sentences

An imperative sentence consists of either a simple or compound imperative statement terminated by a period. An imperative sentence may contain either a GO statement or a STOP RUN statement, which (if present) must be the last statement in the sentence.

Examples:

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

c. Conditional Sentences

A conditional sentence is a conditional statement terminated by a period. Conditional sentences are written in the following format:

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

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

d. Compiler Directing Sentences

A compiler directing statement terminated with a period is a compiler directing sentence. Compiler directing sentences direct a COBOL compiler or processor to take action at compilation time.

Compiler directing sentences may result in inclusion of routines in the object program. They do not directly result in either the transfer or passing of control. The routines themselves, which the compiler directing sentences may have included in the object program, are subject to the same rules for transfer or passing of control as if those routines had been created from procedural sentences only.

Example: ENTER SPECIAL-METHOD-2-ROUTINE.

e. Control Relationship Between Procedures

In COBOL, Imperative and Conditional sentences describe the procedure that is to be accomplished. The sentences are written successively, according to the rules of the REFERENCE FORMAT (Section V), to establish the sequence in which the object program is to execute the procedure.

In the PROCEDURE DIVISION names are used so that one procedure can reference another by naming the procedure to be referenced. In this way, the sequence in which the object program is to be executed may be varied simply by transferring to a named procedure. The name consists of a noun followed by a period, and the name precedes the procedure it names.

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

6. PARAGRAPHS

So that several sentences may be grouped to convey one idea (procedure), paragraphs have been included in COBOL. In writing procedures in accordance with the rules of the PROCEDURE DIVISION (see VII), and the requirements of the REFERENCE FORMAT, it is necessary to begin a paragraph with a name. A paragraph is the smallest grouping of the PROCEDURE DIVISION which is named.

The source programmer will usually place compiler directing sentences in their own paragraphs. Paragraphs comprised of compiler directing sentences are called "Compiler Directing Paragraphs". Paragraphs which contain at least one procedural sentence are called procedural paragraphs.

Compiler directing sentences may also appear in Procedural Paragraphs. Compiler Directing sentences direct the COBOL processor to take action at compilation time. On the other hand, Procedural sentences denote action to be taken by the object program.

Compiler directing sentences may result in inclusion of routines in the object program. They do not directly result in either the transfer or passing of control. The routines themselves, which the compiler directing sentences may have included in the object program, are subject to the same rules for transfer or passing of control as those routines created from procedural sentences.

7. 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. Sections must be named. This is done by writing the procedure-name, the key word SECTION, and a period; the remainder of the line must be left blank. The Procedure Division need not be broken into sections. However, if there is a need for "program segment overlays", the section to be overlaid by another segment must be named and be assigned a priority number. The section-name applies to all paragraphs following it until another section-name is found. It is not required that a program be broken into sections.

8. COBOL DATA ORGANIZATION

1) General

Writing the Procedure Division tells the COBOL compiler how a problem is to be handled. In a computer there must be information coming in from some source, calculations made inside the computer and results given out in some usable form.

The compiler must then be told where information is stored and in what fashion, what is to be done with this information and how the resultant information from the process should appear as output.

The manner in which information is stored, its order and characteristics are described to the COBOL compiler in the DATA DIVISION. The description of the data item, including length, location of decimal point, dollar sign and the relation of this item to other items are specified in the Data Division.

In the Data Division items are given data-names, condition names are specified; in short, most names used in the Procedure Division are described and assigned. The names not described in the Data Division are:

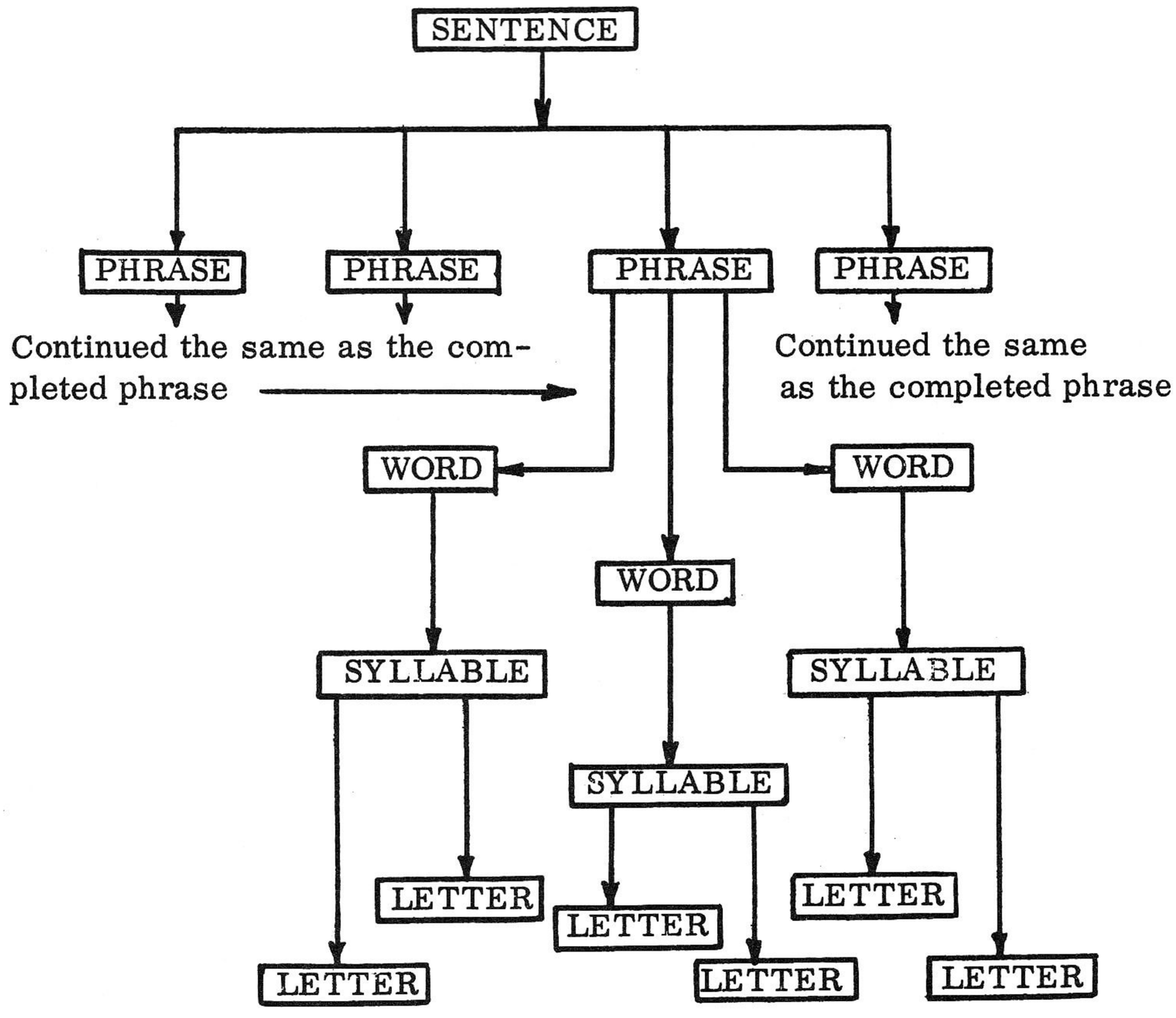
a) Procedure names

b) Special names related to the hardware and defined in the Environment Division.

c) Figurative constants, which have permanent meanings.

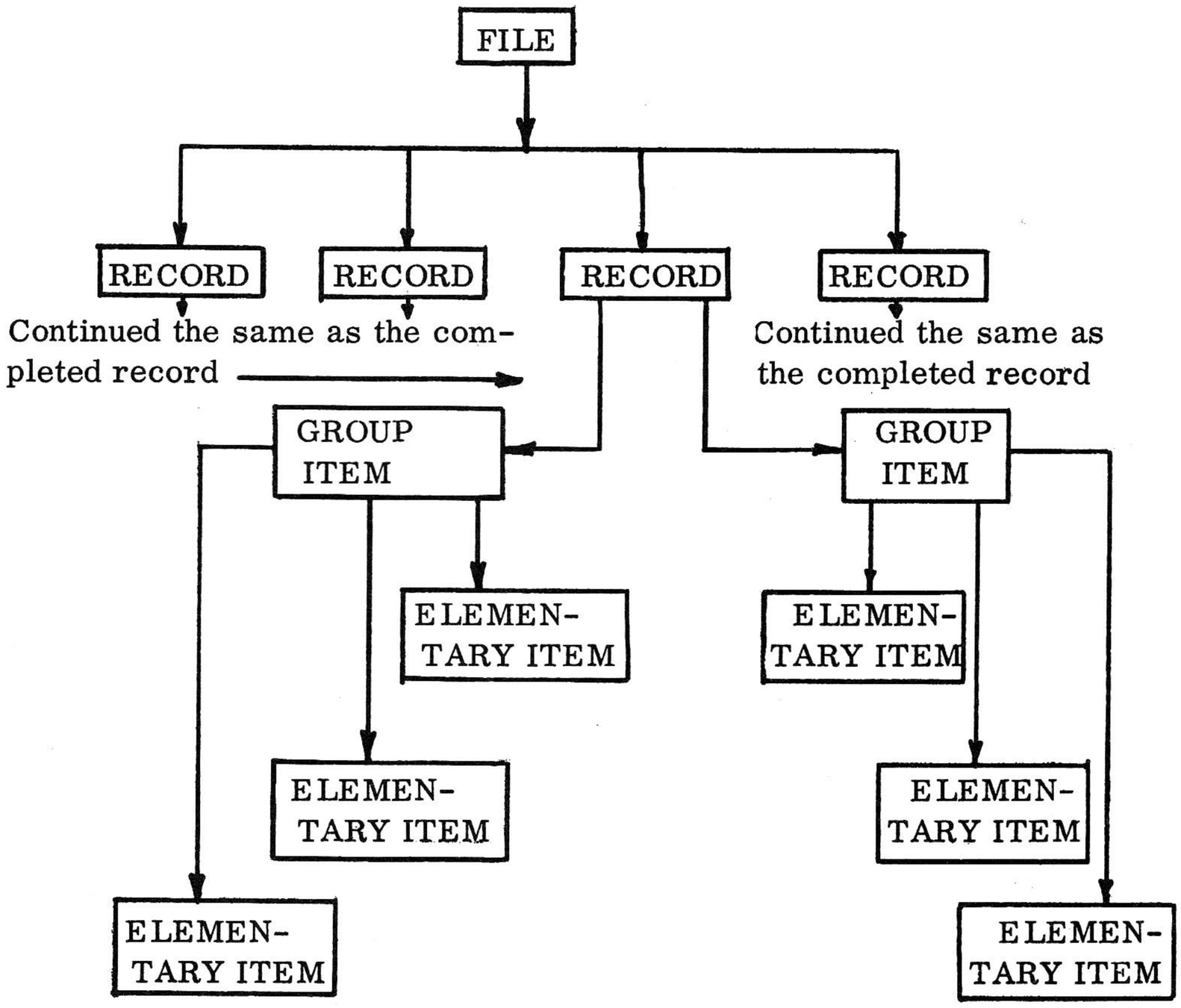
2) Organization of Related Data

To be meaningful, data must be organized in a related pattern. A sentence may be considered an entity, or within a sentence there are phrases, but within phrases there are words, and within words there are syllables, and finally letters.



In a large sense, a sentence can be considered an item, or the phrases can be thought of as a series of items, and words as more series of items.

In COBOL there is a similar pattern of organization for data. The largest entity that COBOL uses is a file. A file then has any number of records, and records contain group items and elementary items.



Each record could be extended as the first one has been.

a) Items

The basic element of data is an elementary item. This is a piece of data which is not further subdivided when it is referenced. If we always referenced NAME, NAME would be an elementary item. If the Procedure Division referenced

FIRST, MIDDLE or LAST (which are parts of the larger entry NAME), then NAME becomes a group item and FIRST, MIDDLE, and LAST would be elementary items.

If there were several classifications in this file named by regions EMPLOYEE-1, EMPLOYEE-2, etc., and each contained the group item NAME, then EMPLOYEE-1 becomes a group item containing within it group items. It is a group of groups.

So elements are combined to form groups which can form groups of groups. An item can then be an element or a group. An element is a unit of data which is never broken into smaller units. For further clarification, see the discussion of Levels in the introduction to the DATA DIVISION, SECTION 6.

b) Records

A record is a group of related data items, and is composed of elementary items and groups of items. A series of similar records is called a File.

A Master File might be made up of a series of Employee Records, one for each employee. All information pertinent to an employee would be contained in his Employee Record. Although the data would vary from employee to employee, the organization within the records would be constant.

Another file might be made up of records, one for each employee, containing only an employee number and items pertaining to the employee's work for one week. In a payroll application, two records would be used for one employee: one from the Master File, and one from the Weekly File.

A RECORD is made available for processing by writing:

READ MASTER-EMPLOYEE for the Master record, or

READ WEEKLY for the weekly record.

It is not possible to read only part of a record or to write only part of a record. Read or write do not necessarily cause a read from a magnetic tape unit or other device; it simply makes a record available to the programmer no matter how the program itself had to get the item.

c) Files

The name file is used in COBOL because it has similarity to the method of storage. A file cabinet contains the records of customers at an automobile agency. The automobile agency might have another cabinet for spare parts. Each of these would be a file. Electronically, the same information would be kept on some quick access storage such as a reel or reels of magnetic tape. In both ways, a cabinet or a reel of magnetic tape, the file would be made up of records.

A file in COBOL is a series of records. In UNIVAC III COBOL, all the records need not have the same format but must be of equal size. When a READ statement provides a record to be processed, it replaces the previous record of that file. If it is necessary in processing the file to have more than one record available at a time, the first record must be read and moved into a work area before the second is read. This can be accomplished in several ways. (SEE READ in the Procedure Division, Section VII.)

The file cabinet had a slot where a label was kept on the front of the drawer to indicate to a clerk the file in the cabinet. Similarly, a file often has a label record to identify the file, usually appearing before the records and another type of label appearing at the end of the file. These labels serve as checking devices to be sure that the program has the file it needs and to assure that the file has been fully read. These functions are most often handled automatically.

A reel of tape may contain one file or one file may extend across several reels; there is no direct relation of a file to a reel of tape.

Two verbs used in handling a file are OPEN and CLOSE. Before a record may be READ, the file must first be "opened", that is made available for processing. Checking the label is usually done in the "opening" process. When all the records that a program needs are used -- usually the entire file --, the file is "closed". The CLOSE verb will write or check the concluding label if present and make the file unavailable for processing until it is again "opened". READ or WRITE cannot be used before an OPEN statement is issued or after a CLOSE statement.

3) Data Division Entries

The Data Division is the place where the programmer specifies to the COBOL compiler the organization of the data used in the program. This division is made of entries, each entry contains one or more independent clauses. The Data Division always has from one to four sections: the File Section, the Common-Storage-Section Working-Storage-Section and the Constant Section, always arranged in the order given.

The File Section contains all entries describing the data coming in or going out of the process. Each file is described by a File Description entry and each record by one or more Record Description entries. The specific entries are described in the Data Division, Section VI.

The Common-Storage-Section contains working storage areas that are used by several parts of a program. These parts of a program would be a Master Program in memory at all times and several subordinate programs (or sections of a program). Each subordinate section may be compiled separately. The Common-Storage-Section has the same kind of entries as the Working-Storage-Section. For a full description of the use of the Common-Storage-Section, see Data Division, Section VI.

The Working-Storage-Section contains Record Description entries specifying work areas needed during the processing of the data. During the computations necessary for handling records, it may be necessary to retain information such as running totals aside from the individual item or it may be necessary to separate an item into its component parts, process and reassemble. For a full description of this entry, see Working Storage Entries in Section VI.

The Constant Section of the Data Division contains all named constants. They are specified in the same way as file records or work areas, by Record Description entries.

4) Levels

Level-numbers are used to specify the relation of item to item, items to groups, groups to groups and on up to files. This organizational format is referred to as the hierarchy of referencing. For a full discussion, see Level-numbers in the introduction to the Data Division, Section VI.

5) Subscripts

a) Data processing makes frequent use of tables of information. COBOL has provided a simple method to access data from tables. This method is called subscripting.

Tables must be described in the Data Division just as any other data. If each item in a table had to be described, it would be quite laborious, particularly for large multi-dimensional tables. Consider a table consisting of sales for each month of the year. The first item would be sales for January, the second sales for February, etc., in calendar sequence. If each month's sales had to be named, this would involve 12 entries and each month's sales could be referred to at any time by its own data-name. A simpler and shorter way to describe the table would be to use the OCCURS clause (see Section VI). The OCCURS clause permits the description of the subsequent item within the table. COBOL provides for a maximum of 3 subscripts (a 3 dimensional table).

b) One Subscript

To subscript, a Record Description entry is written describing the sales data for one month - as a 6-digit integer for example - and state that this item "OCCURS" 12 times. This one entry will describe 12 items, however, they do not have individual data-names and cannot be directly referenced. They have each been named by the one name appearing in the entry describing all 12 items. Describing a table in this manner would require two Record Description entries as follows (this is not a complete Record Description):

```
01  SALES-BY-MONTH
    02  SALES OCCURS 12 TIMES
```

The data-name SALES-BY-MONTH represents all of the 12 months or the entire table.

Now each sales figure can be referred to in the following manner: the sales are arranged by order of the months, thus, the sales for January is the first figure in the table, February is second, etc. Therefore, the sales for January can be referred to as SALES (1) and the sales for February as SALES (2). This is subscripting. Numeric literals or data-names used to locate items in a list or table are "subscripts". Subscripts must always be enclosed in parentheses immediately following the data-name and its qualifiers to be subscripted.

The example above has, in effect, set up a code number for each month; that is, 1 represents January, 2 represents February, and so on for each of the 12 months. This code number could also be named. If an item called MONTH is defined and this can assume different values, under program direction, such as 1, 2, 3, through 12, then MONTH can be used as the subscript for SALES and the value of MONTH specifies which SALES were referenced.

c) Two Subscripts

If the tables of Sales were expanded to include sales by district, for 10 districts, plus a total sales as the 11th district, it would now be a two dimensional table. For example:

JANUARY	FEBRUARY	MARCH	APRIL
DISTRICT 1	DISTRICT 1	DISTRICT 1	DISTRICT 1
DISTRICT 2	DISTRICT 2	DISTRICT 2	DISTRICT 2
DISTRICT 11	DISTRICT 11	DISTRICT 11	DISTRICT 11

Dimension one is across and dimension two is down; that is, to get to District 2 sales for March requires reading across to March and down to District 2. To use subscripts to access March sales for District 2 would require two subscripts and both would be contained within one set of parentheses. The table would be described by a Record Description entry with a data-name for each level of the first item and the OCCURS clause would be used again. For example, these three entries describe all 132 items:

```
01  SALES-BY-MONTH
    02  SALES OCCURS 12 TIMES
        03  DISTRICT-SALES OCCURS 11 TIMES
```

To refer to the sales of a specific district for a specific month, two subscripts are required. For example, to refer to the April Sales for District 4 would mean to refer to the fourth district field of the fourth series of district fields. To refer to total sales for all districts in June would refer to the eleventh district field of the sixth series of district fields, or DISTRICT-SALES (6, 11). If an item is described as DISTRICT and assumes specified values under program control (subscripts being variable must be under program control), it can be used as the subscript. For example, SALES (MONTH, DISTRICT) would be the same as the preceding if MONTH contained the value 6 and DISTRICT contained the value 11.

d) Three Subscripts

As previously stated, COBOL allows for a maximum of three subscripts. This may be illustrated by expanding the SALES-BY-MONTH table so for each month and each district, three department figures are included, one for the whole sale, one for the retail, and one for service. Then the Record Description entries would appear as follows:

```
01 SALES BY MONTH
   02 SALES .... OCCURS 12 TIMES ....
      03 DISTRICT-SALES .... OCCURS 11 TIMES ....
         04 DEPARTMENT-SALES .... OCCURS 3 TIMES ....
```

Here the retail department sales of District 4 for August would be referred to as DEPARTMENT (8, 4, 2), presuming retail sales is the second department.

If desired, all 33 sales fields (11 districts, 3 departments each) of any month can be referenced by using just the major (first) subscript. For example, all sales fields for December can be referenced as SALES (12), or the entire table can be referenced as SALES-BY-MONTH or to reference all departments of July District 9 by DISTRICT-SALES (7, 9).

e) Subscripting Rules

(1) Subscripts must be positive integral values. Zero is not permissible as a subscript value.

(2) Subscripts may be literals or data names.

(3) Subscripts must be enclosed in parentheses and appear immediately following the data-name plus qualifiers of the item being subscripted.

(4) When multi-subscripting is used, all subscripts must be contained within the same set of parentheses and in descending order of inclusiveness from left to right. The subscripts are separated by commas and, as previously defined, a space must follow each comma.

(5) Subscripting can only be used when the Record description of item referred to contains an OCCURS clause.

(6) Subscripting is not restricted to elementary items. The level of inclusion of subscripts for a multi-dimensional table determines the size of the data referenced.

(7) Subscripts must not be subscripted.

(8) If items requiring subscripting are assigned conditional-names, then the conditional-names must also be subscripted when referenced.

f) COBOL Library

(1) General

The COBOL library, called "the library", is a collection of pre-written parts of COBOL programs. At an installation, these parts are placed on the library, usually a tape, because many programs would be using a part exactly as it is. This saves the specifying of the part each time it is used. A record description used by many programs could be written once, placed in the library and copied when needed. The library is made available as input to the COBOL compiler along with the source program.

The library contains entries which may be copied for the Data Division and Procedure Division. For specific details, see the COPY clause in each division.

(2) Entries Associated With Data Division

File Description entries and Record Description entries may be copied from the library. A File Description entry is separate in the library and, if referred to by COPY, only the File Description entry is copied. The Record Description entries of that file will have to be specified or copied apart from the File Description copy. A copy of a Record Description may copy one item or a group of items depending on the Record Description as found in the library.

NOTE: Read carefully the use of COPY for the particular entry when using the library.

LIST OF RESERVED WORDS

The words shown below are an inherent part of the COBOL System. Users should avoid using these words for data or procedure names. Additional words will be added to this list in supplemental reports to reflect the complete list of UNIVAC 1107 COBOL reserved words.

ACCEPT
ADD
ADVANCING
AFTER
ALL
ALPHABETIC
ALPHANUMERIC
ALTER
ALTERNATE
AN
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
BEFORE
BLANK
BLOCK
BY
CARD-FIELDATA
CARD-PUNCH
CARD-PUNCHES
CARD-PUNCH-EIGHTY
CARD-PUNCH-EIGHTIES
CARD-PUNCH-NINETY
CARD-PUNCH-NINETIES
CARD-READER
CARD-READERS
CARD-READER-EIGHTY
CARD-READER-EIGHTIES
CARD-READER-NINETY
CARD-READER-NINETIES
CHARACTER
CHARACTERS
CHECK
CLASS
CLOSE
COMMON-STORAGE
COMPUTATIONAL
CONFIGURATION
CONSOLE
CONSTANT
CONTAINS
CONTROL
COPY
CORRESPONDING
DATA
DATE-COMPILED
DATE-WRITTEN
DEMAND
DEPENDING
DESCENDING
DIAGNOSTIC-FILE
DIGITS
DISC-FILE
DISC-FILES
DISPLAY
DIVIDE
DIVISION
DOLLAR
DRUM
DRUMS
ELSE
END
ENTER
ENVIRONMENT
EQUAL
ERROR
EVERY
EXAMINE
EXIT
FD
FILE
FILE-CONTROL
FILLER
FIRST
FLOAT
FOR
FROM
GIVING
GO
GREATER
ID
IDENTIFICATION
I-O-CONTROL
IF
IN
INPUT
INPUT-OUTPUT
INSTALLATION
INTERNAL
INTO
IS
JUSTIFIED
LABEL
LEADING
LEAVING
LEFT
LESS
LIBRARY
LINES
LINES-AT-BOTTOM
LINES-AT-TOP
LINES-PER-PAGE
LINE-SPACING
LOAD
LOCATION

LOCK
MEMORY
MODE
MONITOR
MOVE
MULTIPLE
MULTIPLY
NEGATIVE
NEXT
NO
NOT
NOTE
NUMERIC
OBJECT-COMPUTER
OCCURS
OF
OMITTED
ON
OPEN
OPTIONAL
OR
OTHERWISE
OUTPUT
PERFORM
PICTURE
PLACES
POINT
POSITION
POSITIVE
PRINTER
PRINTERS
PRINTER-FIELDATA
PROCEDURE
PROCEED
PROGRAM-ID
PROTECT
QUOTE
READ
RECORD
RECORDING
RECORDS
REDEFINES
REEL
REFERENCING
RELEASE
REMARKS
RENAMING
REPLACING
RERUN
RESERVE
RETURN
REWIND
RIGHT
ROUNDED
RUN
SAME
SECTION
SECURITY
SELECT
SENTENCE
SEQUENCED
SIGN
SIGNED
SIZE
SORT-FILE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDBY
STOP
SUBROUTINE
SUBTRACT
SUPERVISOR
SUPPRESS
SYNCHRONIZED
TALLY
TALLYING
TAPE
TAPE-PUNCH
TAPE-READER
THAN
THEN
THROUGH } Equivalent
THRU }
TIME
TIMES
TO
TRANSLATION
UNISERVO
UNISERVOS
UNISERVO-II
UNISERVO-IIS
UNISERVO-III
UNISERVO-IIIS
UNISERVO-IIIC
UNISERVO-IIICS
UNIVAC
UNIVAC-1107
UNTIL
UPON
USAGE
USE
VALUE
VALUES
VARYING
WHEN
WITH
WORD
WORDS
WORKING-STORAGE
WRITE
ZERO
ZEROES
ZEROS

V. COBOL REFERENCE FORMAT

1. GENERAL

The COBOL language must be used very precisely and in accordance with the rules stated. The arrangement and description of the data must also be in precise form if the data is to be usable.

Since the COBOL source program must be entered into the COBOL compiler program which had been loaded into the computer, the source program must also follow a very precise format when it is written. A standard COBOL Reference Format is used for this. The Reference Format prescribes the sequence of the source program and the certain ways in which information must be arranged.

2. PURPOSE OF REFERENCE FORMAT

The reference format has three main purposes.

a. It provides a convenient form for the programmer to use, the COBOL Program Sheet. This sheet helps both the programmer and the person who punches cards from this format to arrange the program in the proper form.

b. It specifies to the COBOL compiler procedure names and other items the compiler needs to create the object program.

c. It gives 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.

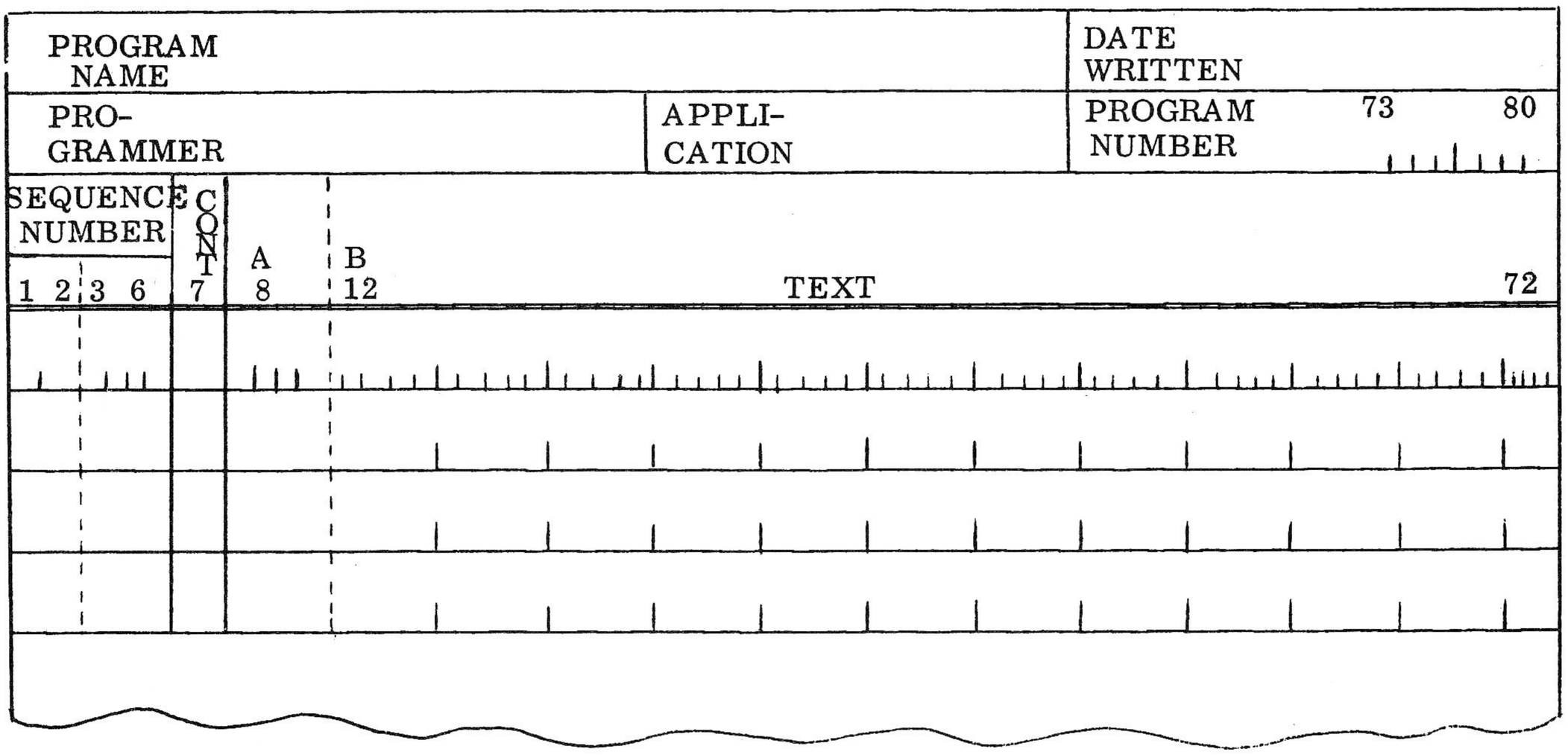
Although this Reference Format is of "free" form in appearance, the rules for using it are precise and take precedence over any other rules stated in this manual about spacing of names, data and other formats.

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

3. USING THE REFERENCE FORMAT

a. General

The layout shown is a diagram of the Reference Format. The form provides for 80 columns of information, reflecting the fact that originally the information from one line is keypunched into an 80 column card.



Columns 1 through 6 are used for sequence numbers. Sequence numbers are an aid in corrections and changes. The sequence number is purely numeric. Letters of the alphabet and special characters are not allowable. For the use of correction cards for COBOL programs, see Corrections, Section X.

The compiler will test for sequence numbers out of sequence and make note of these. The compiler will assign line numbers, which will replace any sequence numbers. When using the COPY---FROM LIBRARY clause in any DIVISION, sequence numbers are not called in.

Column 7 is used as a continuation indicator.

Many paragraphs and entries need more than one line. If the break in the line occurs in the middle of a word or literal, the continuation of the word or literal must be signalled by a hyphen in Column 7 of the next line. Unless a hyphen appears, the word or literal will be considered to have ended right there. If the hyphen is used, then the first character of the hyphened line will be considred a continuation of the word or literal. This is the only way to carry a word or literal over from one line to the next without a break in the word or literal.

If a non-numeric literal is carried over, there must be a hyphen in Column 7 of the next line and the continuation of the non-numeric literal must start with a quotation mark.

Whether or not the continuation indicator is necessary, the rules specified for indenting continued items must be followed.

Columns 8 through 72 or 80 are used for the data (in description entries) and the instructions (procedures) that go into the computer. Two margins are necessary for aligning the data in these columns: Margin A, to the left of Column 8, and Margin B, to the left of Column 12. An item lined up with Margin A has its first character in Column 8. An item lined up with Margin B has its first character in Column 12.

The names of divisions, sections, paragraphs and the main entries of the Data Division are placed at Margin A. Margin B is used for subordinate items and continuations of items from one line to another.

Columns 73 through 80 may be used at the option of an installation in any fashion that the installation may desire. All characters of the 1107 COBOL character set may be used, including the blank; the information in these columns is ignored by the compiler, but is printed in the listing.

b. 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 position 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 under position B on the next line. Any word or numeric literal can be split over two lines by placing a hyphen in the seventh character position of the second line. A non-numeric literal can also be split over two lines by placing a hyphen in the seventh character position of the second line, and a quotation mark at the start of the continuation of the non-numeric literal.

A sample format for this division is as follows:

```
L          A       B                                                  R

000100     IDENTIFICATION DIVISION.
000200     PROGRAM-ID.      PAYROLL.
000300     AUTHOR.     JOHN DOE.
000600     DATE-WRITTEN.          APRIL 5  1960.
001100     REMARKS.    INPUT FROM RUN 4 AND OUTPUT
001101                 TO RUN 25.  THIS PROGRAM
001102                 PROCESSES SALARIED EMPLOYEES
001103                 ONLY.
```

c. Rules for the Environment Division

The Reference Format for the ENVIRONMENT DIVISION is the same as that of the IDENTIFICATION DIVISION. In addition to fixed paragraph-names, there are also fixed Section-names. The name of the division, and the names of the Sections and paragraphs within it start under Position A. The first line of the division consists of its name, followed by a period; i.e.,

ENVIRONMENT DIVISION.

Section names, like the division name, appear as a single entry on a line by themselves. (See example that follows.)

The rules for continuation of sentences and the splitting of words or literals over two lines are the same as those in the IDENTIFICATION DIVISION. The I-O-CONTROL and the FILE-CONTROL paragraphs are each composed of several sentences, whereas the other paragraphs are each composed of one sentence only. The following is an example of a possible ENVIRONMENT DIVISION format.

```
L        A    B                                                    R
000100   ENVIRONMENT DIVISION.
000200   CONFIGURATION SECTION.
000300   SOURCE-COMPUTER.   Computer-name....
000400   OBJECT-COMPUTER.  Computer-name......
001100   SPECIAL-NAMES.   Hardware-name........
001200   INPUT-OUTPUT SECTION.
001300   FILE-CONTROL.  SELECT  File-name-1....
001400          SELECT File-name-2.....SELECT.....
001500   I-O-CONTROL.  APPLY.........
```

d. Rules for the Data Division

The basic unit in the DATA DIVISION is an entry. Each entry begins with a level number, followed by the name of a data item, and a sequence of independent clauses descriptive of the item. The same format rules apply to the File and Record Description portions, as well as to the Working Storage, Common and Constant Sections of the DATA DIVISION.

The user may choose whether to left-justify successive entries or indent them according to level number.

If he chooses to indent all successive levels according to level number, he is restricted by the physical medium to a limited number of levels, but he obtains the advantage of displaying graphically the hierarchical structure of data. To combine the advantage of graphical display through indentation and the use of larger number of levels, the user may indent some levels and not others. The entries in the Reference listing need be indented only if the input was indented. Under no circumstance does indentation affect the magnitude of a level number.

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

```
L       A    B                                          R
XXXXXX  NN   data-name-1 ..... ... ........
XXXXXX       NN data-name-2 .....  .......
XXXXXX            NN data-name-3 ..............
```

Using the same representation, the specific format for the DATA DIVISION, not using indentation at all, is:

```
L       A    B                                          R
XXXXXX  NN   data-name-1 ....................
XXXXXX  NN   data-name-2 ....................
XXXXXX  NN   data-name-3 ....................
```

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

DATA DIVISION.

The sequence number appears at the left as in the format for the IDENTIFICATION DIVISION. The first level number starts under Position A. If a single entry requires more than one line, the left margin for each line is the same, namely, the position under the first letter of the data-name. The rules for the splitting of words or literals over two lines are the same as those in the IDENTIFICATION DIVISION.

When level numbers are to be indented, each new level number must begin 4 spaces to the right of the starting position of the previous level number. One or more spaces must be left between the level number and the data-name. The maximum number of indented levels will be determined by the width of the physical medium (printed page, punched cards, etc.)

An example of the DATA DIVISION reference format is:

```
L      A    B                                                    R
000010 DATA DIVISION
000100 FD   MASTER-PAYROLL LABEL RECORDS ARE
000200      STANDARD DATA RECORDS ARE MASTER-
000300-     PAY SEQUENCED ON BADGE-NUMBER.
000400 01   MASTER-PAY SIZE IS 180 CHARACT
000500-     ERS  CLASS IS ALPHANUMERIC.
000600      02  BADGE-NUMBER SIZE IS 12 CH
000700-         ARACTERS  PICTURE IS
000800          AAAXXX999999.
000900      02  DATE SIZE IS 6 CHARACTERS
001000          CLASS IS NUMERIC.
001100      03  MONTH SIZE IS 2 CHARACTERS.
001200      03  DAY SIZE IS 2 CHARACTERS.
001300      03  YEAR SIZE IS 2 CHARACTERS.
001400      02  GROSS-PAY SIZE IS 6 CHARAC
001500-         TERS  PICTURE IS 0999V99.
```

e. Rules for the Procedure Division

The Reference Format for the PROCEDURE DIVISION is as follows:

The first line of the division consists of its name, followed by a period, starting under Position A, as follows:

PROCEDURE DIVISION.

If a section has been designated, the Section-name starts under Position A, followed by a space, the word SECTION, a space followed by a priority number (when used), a period and the rest of the line blank.

A paragraph consists of one or more successive sentences, the first of which must be preceded by a paragraph-name. The name starts under Position A and is followed immediately by a period.

A new paragraph is determined by the appearance of another paragraph-name. Note that a paragraph may consist of only a single sentence.

Any sentence which occupies more than one line must be continued by starting under Position B on the next line. 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 start the word or literal on the next line.

An example of the PROCEDURE DIVISION format is:

```
L      A    B
000100 PROCEDURE DIVISION.
000200 COMPUTATIONS SECTION.
000300 UPDATE-MASTER.
000400         MOVE ADJUSTED-PAY TO NET-PAY.
000500         ADD GROSS-PAY TO GROSS-YEAR-TO-DATE.
000600         WRITE UPDATED-MASTER-PAY.
000700         READ MASTER-PAYROLL-RECORD  AT END
000800         GO TO MASTER-PAYROLL-END.
000900         GO TO INITIALIZATION-ROUTINE.
001000 ERROR-ROUTINE.
001100         DISPLAY "ERROR FOR" TIME-CARD-
001200-        NUMBER UPON PRINTER.
001300         READ TIME-CARD-RECORD  AT END GO
001400         TO TIME-CARD-END.
001500         GO TO INITIALIZATION-ROUTINE.
```

VI. DATA DIVISION

1. GENERAL DESCRIPTION

Each file, record, and data item is described for use in a program by writing data description entries in the Data Division of the program. Data processed falls into three categories: (1) data which is contained in files and enters or leaves the internal memory of the computer from a specified area(s); (2) data which is developed internally and placed into intermediate or working storage, and (3) constants which are defined by the user. Tables may fall into any of the named categories.

Every data-item referred to in the Procedure Division must be described in the Data Division, excepting the special name TALLY. Figurative constants and literals used in procedure statements are not listed in the Data Division. Items and records are described by Record Descriptions entries, and files are described by File Description entries. This section is devoted to the details of writing the Data Division of a COBOL Program.

The approach taken in defining file information is to distinguish between the physical aspects of the file (i.e., the File Description) and the conceptual characteristics of the data contained therein (i.e., the Record Description). Physical aspects refer to the mode in which the file is recorded, the grouping of logical records within the physical limitations of the file-medium, the means by which the file can be identified, etc. Conceptual characteristics refer to the explicit definition of each logical entity within the file itself.

For purposes of processing, the contents of a file are divided into logical records. (A logical record is any related consecutive set of information.) Because of the logical record concept, a COBOL user may relate the problem definition to the steps taken to manually act upon the record. When a problem deals with a single input file, the program treats this file one record at a time, and generally in sequence. Therefore, after the file is opened, the first logical record is read and processed in some manner prior to returning to the read operation for the second record of the same file. Some of the various situations that exist will point out some of the logical controls the problem solution may require using this technique.

a. Often the records to be processed are restricted to a certain portion of the records in the file depending upon some logical portion of the record. Thus, the problem solution would first take a look at the logical portion (key) of the record to find out if the record was one to be processed.

b. In a situation where the logical records are supposed to be in some order, but manual intervention may introduce out of sequence conditions, the key of the second and succeeding records are tested against the key of the previous record to insure that the logical records are in sequence (b is greater than a). Group controls are also provided in making similar checks and initiating control change processes when keys are not equal.

c. Where many of the logical records have identical keys, the problem solution may include a summarizing of logical records. This may be accomplished by delaying the main processing and saving the current record until logical records which follow are checked for identical keys, and summarized until a new key is found. Summarizing may be accomplished in this manner on input, output, and on sequenced or random logical records.

In every case the problem solution may only deal with one logical record at a time. Some users of COBOL with knowledge of the manner in which groups of logical records are batched for inputs and outputs on some computers are thus confused by the difference between logical and physical records. Unit record equipment shares this problem. The physical media, an 80 or 90 column card, may contain a portion of the whole logical record in which case several cards are required to represent a single logical record. In another case the physical record may contain two or more logical records.

The problem solution written in COBOL defines the relationship between a physical record requirement (card, tape block, print line, etc.) and the logical record but must define the solution in terms of logical records.

Each input and output file is assumed to have a single area in memory which is large enough to contain the largest logical record which may exist in the file. This area may be part of the area used to store the physical record, or some separate area. In any event, files are described and COBOL problem solutions are stated, as though the first logical record of a file

becomes available when the first READ for that file is executed. All subsequent records may only be addressed after they have been "read" into the single unique area. Two or more logical records from the same file cannot be addressed by a program unless the program provides for the saving of the records involved in some other place in storage. This other place may be in Working-Storage or an output file record area depending upon the problem. Tables maintained as external files (card, tape, paper tape, etc.) are sometimes read into storage one record at a time and moved to a Working-storage.

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

File Descriptions and Record Descriptions may be stored in a library.

2. ORGANIZATION

The Data Division is composed of four sections, each of which consists entirely of entries rather than paragraphs and sentences. The four sections are: (1) File Section, (2) Common-Storage Section, (3) Working-Storage Section, and (4) Constant Section, written in that order within the division. If any one of the sections is not required in a particular program, it may be omitted entirely.

The Data Division begins with the header DATA DIVISION. Each of the sections also begins with a header; the name of the section is followed by the word SECTION and a period as follows:

```
FILE SECTION.
COMMON-STORAGE SECTION.
WORKING-STORAGE SECTION.
CONSTANT SECTION.
```

The FILE SECTION contains two elements: (1) descriptions of files and (2) descriptions of records. The File Section contains File Descriptions and Record Descriptions for both label records and data records. These two kinds of records are defined in the same manner; however, because the input/output system of the object program must perform special operations on label records, fixed names have been assigned to certain label record items on which specified operations must be performed. All Record Description entries pertaining to the label records and data records of a file must immediately follow the File Description entry for that file. That is, the File Description entry represents the highest level of organization in the FILE Section.

The COMMON-STORAGE, WORKING-STORAGE and CONSTANT Sections consist solely of Record Descriptions, and unrelated Record Description entries.

DATA DIVISION SAMPLE ENTRY

An example of three sections of the Data Division, as they would appear in a COBOL program, follows:

```
L      A      B                                                                R
1      8      12                                                               73
200100 DATA DIVISION.                                                          7543004
200200 FILE SECTION.                                                           7543004
200300 FD  EDITED-SHIPMENTS  LABEL RECORD IS STANDARD DATA RECORD IS            7543004
200400     DETAIL-SHIPMENTS  BLOCK CONTAINS 20 RECORDS                         7543004
200500     VALUE OF ID IS "7543004A".                                          7543004
200700 01  DETAIL-SHIPMENTS.                                                   7543004
200800     02  A-FIELD CLASS IS NUMERIC USAGE IS COMPUTATIONAL.                 7543004
200900         03  CONTROL WORD.                                               7543004
201000         04  WORKS SIZE IS 2 COMPUTATIONAL DIGITS.                       7543004
201100             88  TIN-MILL VALUE IS 06.                                   7543004
201300         05  FIRST-DIGIT PICTURE IS H9.                                  7543004
201400         05  SECOND-DIGIT PICTURE IS H9.                                 7543004
201500         04  UNIFIED-PRODUCT-SALES-CODE.                                 7543004
201600         05  PROD-CODE SIZE IS 1 CHARACTERS.                             7543004
```

COMMON-STORAGE SECTION.

(Not used in this example but would appear here and be similar in form and content to WORKING-STORAGE SECTION.)

```
350100 WORKING-STORAGE SECTION.                                                7543004
350200  77  VALUE-COST-DIFFERENCE PICTURE IS SH9(7)99 VALUE IS ZERO.           7543004
350400  77  CTR PICTURE IS H999 SYNCHRONIZED RIGHT VALUE IS ZEROS.             7543004
400100  01  HEADING.                                                          7543004
400200      02  HDG-PG-CTRL SYNCHRONIZED RIGHT PICTURE IS H99 VALUE IS 1.     7543004
400400      02  FILLER SIZE IS 54 AN CHARACTERS.                              7543004
400500      02  BODY PICTURE IS X(29) VALUE IS "SALES STATISTICAL DATA RE      7543004
400600-         PORT".                                                         7543004

500100 CONSTANT SECTION.                                                       7543004
500500 77  TIN-MILL-WKS  PICTURE IS H99  VALUE IS 06.                         7543004
510100 01  TABLE PICTURE IS H9(16) VALUE IS 6432160804020100.                 7543004
```

3. STRUCTURE

The Data Division of the source program begins with the following header:

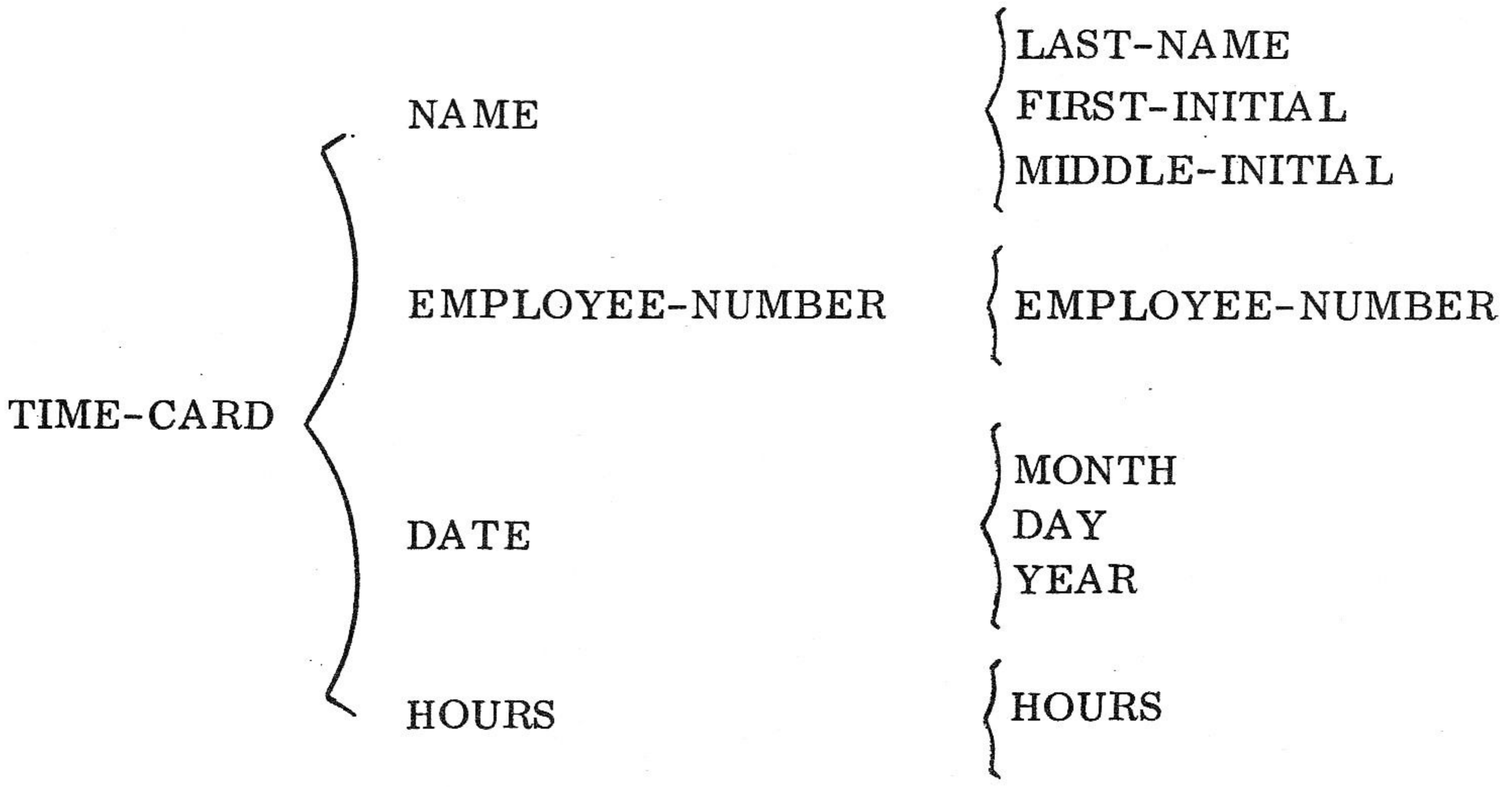
DATA DIVISION.

An entry consists of a level-number, a data-name, and generally one or more independent clauses. (The clauses may be written in any sequence unless the entry format specified otherwise.) An entry is always terminated by a period. A File Description consists of a single entry; however, a Record Description consists of one or more entries.

A File Description entry is identified by the level-indicator FD in place of a level-number. The data-name immediately following the level-indicator is called the "subject" of the entry.

a. Concept of Levels

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



The most basic subdivisions of a record, that is, those not further subdivided, are called elementary items; consequently, a record is said to consist of a sequence of elementary items.

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

A system of level numbers is employed in COBOL to show the organization of elementary items and groups. Level numbers start at 01 for records, since records are the most inclusive groups possible. Less inclusive groups are assigned higher (not necessarily successive) level numbers not greater in value than 49. (NOTE: There are "special" level numbers, 77 and 88, discussed below, which are exceptions to this rule.) Separate entries are written in the source program for each level. This is done by grouping the entries describing the related items and assigning a level-number to each item. If DATE were assigned a level number of 05 and MONTH, DAY, YEAR and STATE were each assigned a level-number of 06, this would indicate that the last four were of a lower level than DATE. In general, the higher the numeric value of the level-number, the lower the hierarchical level of the item; the number denoting the record level is 01; the lowest level is 49. (The level-numbers 77 and 88 are special cases which will be explained later.) The file level is indicated by the special level indicator FD in place of a level-number.

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

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

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

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

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

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

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

The description of a record in the File Section of the Data Division will consist of a series of Record Description entries, one for each item. The first item of a Record Description entry is always a level-number; the second is the data-name which will be assigned to the item. These are followed by several clauses which completely describe the item.

The entries might be arranged as follows:

```
01    EMPLOYEE-RECORD ....
      02    NAME ....
            04  LAST ....
            04  FIRST ....
            04  MIDDLE ....
      02    MAN-NUMBER ....
      02    ADDRESS-A
            05  STREET ....
            05  CITY ....
            05  ZONE ....
            05  STATE ....
```

In the above example, NAME is assigned the level 02. Immediately following are entries describing LAST, FIRST, and MIDDLE. These three items are identified as being part of the group item NAME because:

1) They follow group item NAME and another group item with a level-number equal to or numerically lower than that of NAME has not intervened.

2) They have level-numbers numerically higher than that of group item NAME.

The principal rules for assigning level-numbers are as follows:

1) The level 01 is reserved exclusively for identifying a record.

2) Any item will be contained in a preceding item, if all of the following conditions are met:

   a) It has been assigned a numerically higher level-number.

   b) An entry with a level-number numerically equal to or lower does not occur between the entries. Thus, even though STATE has a higher level-number than MIDDLE, it is not part of MIDDLE, because entries with level-numbers lower than 04 occur between STATE and MIDDLE.

c) The entry to be contained must follow the containing entry.

3) Level-numbers need not be assigned consecutively.

4) When an entry is to have a numerically lower level-number than the one immediately preceding it, the level-number must be chosen from the level-numbers of the groups which include the preceding item. Thus, ADDRESS must be assigned level 02, because that is the only level-number assigned to a group that contains the preceding item, MAN-NUMBER.

Two types of data exist for which there is no true concept of level; namely, non-contiguous constants and working-storage items and condition-names.

Non-contiguous constants and non-contiguous working-storage items which bear no relationship to one another and which may not be further subdivided, have been assigned the special level number 77.

Entries which specify condition-names, to be associated with a particular value of an item, and which do not themselves introduce data, have been assigned the special level number 88.

The level-number 77 is used when describing constants or 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 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; the special level-number 77 is used to denote these independent items.

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 an item may assume. Thus, it is given the special level-number 88. Condition-names are assigned by writing an entry for the item of data itself, immediately followed by an entry for each condition-name to be associated with the item.

b. Data Reference Format

The basic unit in the Data Division is an entry and the user may choose whether to left-justify successive entries or indent them according to level number.

If the user chooses to indent all successive levels according to level number, he is restricted by the physical medium to a limited number of levels, but he obtains the advantage of displaying graphically the hierarchical structure of data. To combine the advantage of graphical display through indentation and the use of larger number of levels, it is suggested that a combination of the two be employed; i.e., indent only through the 03 level. This is left to the discretion of the user. Under no circumstance does indentation affect the magnitude of a level number.

The first line, regardless of whether or not indentation is used, consists of DATA DIVISION. Columns 1 through 6 are used for the sequence number. The sequence number is deleted by the compiler and a line number is inserted instead. The sequence number is purely for the convenience of the 1107 COBOL programmer.

The first level-number (01) starts position A (Column 8). The FD level-indicator and level-number 77 also start under position A (Column 8). The FD or level-numbers 01 and 77 are followed by two spaces, then the data-name under position B (Column 12). Each clause of the entry is separated from preceding name or clause by one or more spaces. The last clause is followed by a period. If a single entry requires more than one line, the left margin for each line starts under the first letter of the data name; i.e., position B (Column 12). The rules for splitting of words or literals over two lines are the same as those discussed in the REFERENCE FORMAT (SECTION V).

If level-numbers are indented, each new level-number must begin four spaces to the right of the starting position of the previous level-number. One or more spaces are then to be left between the level-number and the data-name.

Please note the following example:

```
1        8    12    16
250500   01   ERRORS CLASS IS AN.
250600        02   PG-CTRL COPY PG-CTRL IN EXCEPTION.
250700        02   FILLER PICTURE IS X(35).
250800        02   GRADE.
250900             03   PRE PICTURE IS X.
251000             03   REST PICTURE IS XXX.
```

It can be determined from this example that the 01 level starts at Column 8 and the data-name "ERRORS" starts at Column 12, thus, leaving two spaces between the level-number and the data-name. The same determination can be made at the 02 and 03 levels as shown. Please note that each new level-number begins 4 spaces to the right of its previous level-number; i.e., 02 starts at Column 12 (8 + 4); 03 starts at 16 (12 + 4).

4. FILE DESCRIPTION ENTRY (FILE SECTION)

A File Description entry contains information pertaining to the physical aspects of a file and must be written for each file processed by a program. In general, it may include: (1) the manner in which the data is recorded on the file, (2) the size of the logical and physical records, (3) the name and value of the label record contained in the file, (4) the names of data records which comprise the file and (5) the keys on which the data records are sequenced.

A Complete series of Record Description entries describing every item in a file must immediately follow the File Description entry for that file. The listing of data and label record names in a File Description entry serves as a cross-reference between the file and the records in the file.

a. Entry Formats -- General

A File Description entry consists of a level indicator, a file-name, and a series of independent clauses which define the physical and logical characteristics of the file. The mnemonic level indicator FD is used to identify the start of a File Description entry, and distinguishes this entry from those associated with a Record Description.

b. Entry Formats - Specific

The individual clause format are arranged in this section in an alphabetic order, whereas the clauses are shown in the recommended order in the "Complete Entry" which follows.

1) COMPLETE ENTRY

The function of the File Description complete entry is to furnish information concerning the physical structure, identification and record descriptions pertaining to a given file. The complete entry format follows:

Option 1

FD file-name COPY library-name.

Option 2

FD file-name

[RECORDING MODE IS {CARD-FIELDATA / PRINTER-FIELDATA / INTERNAL}]

[BLOCK CONTAINS integer-1 {RECORDS / CHARACTERS}]

[RECORD CONTAINS integer-2 CHARACTERS] LABEL RECORD(S) [ARE / IS] {STANDARD / OMITTED}

[VALUE OF {IDENTIFICATION / ID} IS {data-name-1 / literal-1} [, DATE-WRITTEN IS literal-2]

[, LINES-PER-PAGE IS literal-3] [, LINES-AT-TOP IS literal-4]

[, LINES-AT-BOTTOM IS literal-5] [, LINE-SPACING IS literal-6]]

{DATA RECORDS ARE / DATA RECORD IS} data-name-2 [data-name-3 ...]

[SEQUENCED ON data-name-4 {ASCENDING / DESCENDING}][, data-name-5 {ASCENDING / DESCENDING}] ...

Option 1 is used when the library contains the entire File Description entry.

The level indicator FD identifies the beginning of the File Description entry and precedes the file name assigned by the programmer; e.g.,

FD EDITED-SHIPMENTS ....

The clauses which follow the data-name are optional in some cases as will be explained in the following specific formats.

The File Description entry is terminated by a period.

2) BLOCK SIZE

The function of the BLOCK clause is to specify the size of a physical record; i.e., the size of the block. Records in a file may be grouped physically in blocks to conserve external storage space; e.g., the length of magnetic tape required to store a file. This physical grouping does not affect the logic of the program. In many applications, however, the amount of storage conserved as a result of blocking may amount to one or more reels of tape. In such cases, the running time of the object program is reduced significantly. A block cannot contain less than one record; it must consist of one or more entire records. That is, a block and a record may be equal in size, or several records may be contained in a block. The general form of the clause specifying the size of a block is shown below:

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

Integer-1 must be an unsigned (positive) numeric literal. Integer-1 must be a multiple of 6 when the CHARACTER option is used.

This clause is required except when:

> A physical record contains one and only one complete logical record.

If the RECORDS option is used in describing a file whose records are of variable size, the compiler will consider all records to be the size of the largest record.

When the CHARACTERS option is used, the physical record size is specified in terms of the number of standard characters contained within the physical record.

The word CHARACTERS within the BLOCK clause is an optional word. Whenever the key word RECORDS is not specifically written in the BLOCK clause, integer-1 represents CHARACTERS. If neither RECORDS nor CHARACTERS is written, integer-1 is assumed to be the number of characters in the block.

For peripherals, this phrase may be omitted. The block sizes are:

HSP	128 standard data format characters
80-column card file	80 or 160 standard data format characters

Note: Any beginning and/or ending block indicators required for the input/output system will be generated by the compiler and be of no concern to the programmer.

3) COPY

The function of the COPY clause in the FD entry is to obtain a file description entry from a library. This clause is used by the processor to incorporate a prewritten File Description entry into the Data Division of a program. When this clause is used, the compiler obtains the specified entry from the library and places it in the program. Since the entire File Description entry is copied from the library, the COPY clause must appear as the only clause in the entry when this option is used. The general form of the complete File Description containing a COPY clause is:

FD file-name COPY library-name.

During compilation, the COPY clause is replaced by the sequence of clauses within the library name entry. Thus, the level indicator (FD), and the file-name which precede the COPY clause will replace the level indicator and file-name appearing within the library-name entry.

Example: Suppose the following FD entry exists in the library:

FD EDITED-SHIPMENTS LABEL RECORD IS .....

This entry could be called from the library by the following entry:

FD MASTER-EDITED-SHIPMENTS COPY EDITED-SHIPMENTS.

The File Description entry, as it is placed in the source program, would appear as follows:

FD MASTER-EDITED-SHIPMENTS LABEL RECORD IS .....

It should be noted that this COPY clause will copy only a File Description entry, not the associated Record Description entries.

4) DATA RECORDS

The function of the DATA RECORDS clause is to allow the processor to cross-reference the File Description entry and the individual Record Description entires of each record whenever necessary. The general form of the clause is:

DATA RECORD IS
DATA RECORDS ARE data-name-3 [data-name-4...]

This clause is required in every File Description entry. Data-name-3, data-name-4, etc., must each be the subject of a Record Description entry with a level-number of 01. Subscripting of these data-names is not permitted, and qualification in this entry is not necessary since they are implicitly qualified by the file-name of this entry. All subordinate records must be named in a DATA RECORD (S) clause. Failure to do so will be treated by the compiler as a correctable error.

It must be remembered that no two records of the same file are made available for processing at the same time. In other words, if one record is read from a file and then another record is read from the same file, the second record replaces the first record. Two possible ways to save the first record could be either to move it to a work area, using the READ verb with the INTO option, or to use the MOVE verb (as explained in SECTION VII).

Since the compiler will process all data record descriptions which follow the FD level entry, any discrepancies between the data record name in this clause and the 01 level entries which follow will cause this clause to be ignored by the compiler. Refer to the "COMPLETE ENTRY" for an example of the DATA RECORDS clause used in the FD entry.

5) LABELS

The function of the LABEL RECORD clause is to allow the compiler to cross-reference the descriptions of a label record with its associated file; i.e., to identify a file or the reels of a file.

The UNIVAC 1107 system has its own standard format for a label record. To specify whether a label record is present in an incoming file, or should be included in an output file, the following clause is used in the File Description entry:

$$\left\{\underline{\text{LABEL RECORD(S)}}\right\} \begin{matrix}\text{IS} \\ \text{ARE}\end{matrix} \left\{\begin{matrix}\underline{\text{STANDARD}} \\ \underline{\text{OMITTED}}\end{matrix}\right\}$$

This clause is required in every File Description entry. When a file contains no beginning or ending, tape or file label, the word OMITTED must be used.

When describing an output file (a file being created by the program), the programmer may specify whether a label is to be present or not, according to his wishes. However, when describing an input file, if a label is present in the file, the programmer must write LABEL RECORD IS STANDARD. Thus, it is illegal to attempt to ignore an existing label record by writing LABEL RECORD IS OMITTED.

The STANDARD option is required for the magnetic tape files and implies the use of the UNIVAC 1107 conventional label and sentinel formats. Completely automatic label and sentinel handling will be provided for this file. No Record Description entry is permitted for the label record but the VALUE OF ID clause in this entry should be specified. The value specified is the thirty character maximum alphanumeric file identification.

The STANDARD UNISERVO III label record will be specified later.

Any additional labels that may be applied to magnetic tape files must be defined and described as DATA RECORDS (rather than LABEL RECORD) and processed through the program's normal procedures.

The OMITTED option is required for peripheral equipment and implies labels described as DATA RECORDS (in Note 4 above) and handled under programmer control.

If the file being described is a file assigned to a peripheral device (card reader, card punch or UNISERVO II) the OMITTED option must be used; e.g., LABEL RECORD IS OMITTED. The "label" (if desired) is described in a DATA RECORD entry. It is the responsibility of the programmer to handle this "label" record in his own fashion.

6) RECORD CONTAINS

The function of the RECORD CONTAINS is to specify the size of data records. The general form of the clause is as follows:

[RECORD CONTAINS integer-2 CHARACTERS]

Integer-2 must be an unsigned (positive) numeric literal.

The size of each data record is completely defined with the Record Description entries, therefore this clause is not required. Any discrepancy between this clause and the descriptions of the data records which follow will cause the compiler to ignore this clause. The Record will contain the number of characters as defined in their descriptions.

If this clause is written, however, integer-2 must be a multiple of 6-bit characters which will produce an integral number of computer words.

Magnetic tape files must have an integral number of records in a block.

For other peripherals, the record sizes are:

High Speed Printer	128 standard data format characters
80-column card file	* 80 standard data format characters
	*160 standard data format characters

* depending on whether or not automatic translation is specified.

Any discrepancy between this clause and the descriptions of the data records which follow will cause the compiler to ignore this clause.

7) RECORDING MODE

The function of the RECORDING MODE clause is to specify the bit configuration of individual files. The general form of the clause is:

$$\left[\underline{\text{RECORDING}}\ \text{MODE IS}\left\{\begin{matrix}\underline{\text{PRINTER-FIELDATA}}\\ \underline{\text{CARD-FIELDATA}}\\ \underline{\text{INTERNAL}}\end{matrix}\right\}\right]$$

If RECORDING MODE IS PRINTER-FIELDATA, the compiler will generate a test to make sure that the record is 128 characters or less in length. If the record is longer than 128 characters, a diagnostic message will be given, but the record will be handled in its entirety. Of course, only 128 characters can be printed on a line, and therefore not all of the over-long record will be printed.

If RECORDING MODE IS CARD-FIELDATA, a test is generated to determine that the record is 80-characters or less in length, and a diagnostic is given if the record is too long. If the record is too long, it will be handled in the same way as the printer record already mentioned. Material being read or punched will be in decimal (Standard Data) format.

If neither of the above clauses is used, and if there is no other indication from either the FILE-CONTROL or I-O-CONTROL paragraphs, RECORDING MODE IS INTERNAL is implied.

Whether it is implied, or directly stated, RECORDING MODE IS INTERNAL provides that the tests generated for PRINTER-FIELDATA or CARD-FIELDATA will not occur. The field will be in the format specified in the data description, by the CLASS or USAGE clause.

If the APPLY TRANSLATION clause is used in the I-O-CONTROL paragraph, RECORDING MODE IS

CARD-FIELDATA is implied. The SELECT file-name ASSIGN TO hardware-name clause in the FILE-CONTROL paragraph will imply RECORDING MODE IS CARD-FIELDATA or PRINTER-FIELDATA. What it amounts to is that the RECORDING MODE clause is not necessary, but may be used, if desired. It may be used in conjunction with the other clauses, but if it is, it should be in agreement.

8) SEQUENCED

The function of the SEQUENCED clause is to indicate the keys on which data records are sequenced. The general form of the clause is as follows:

```
[SEQUENCED ON data-name-4 {ASCENDING  } [data-name-5
                          {DESCENDING }

 {ASCENDING  }] [data-name ...]]
 {DESCENDING }
```

Data-name-4 represents the major key, data-name-5 represents the next highest key, etc.

This clause does not imply an automatic sequence check at object time.

The SEQUENCED clause is required for specifying the keys for Sorting.

When the File Name has been assigned to SORT-FILE in File Control Section (See Section VIII), the SEQUENCED clause is used to specify the sorting keys of the Record Description (See SORTING in SPECIAL FEATURES, SECTION X).

The word ASCENDING is an optional word. Whenever the key word DESCENDING is not specifically written, the sequence will be ascending.

Data-name-4, data-name-5, etc., are the names of the sorting keys in one record in the file. If there is more than one record in the file, the sorting keys of the other record, or records, need not have the same names as data-name-4, data-name-5, etc., but should occupy the same relative positions within the file or files. If sorting keys of differing records are assigned the same data-names, qualification will be necessary whenever they are referred to, throughout the COBOL program.

9) VALUE

The function of the VALUE clause in the FD entry is to specify the file identification in the label of a standard magnetic tape file, to provide for dating a file, and to define page layout for printing. The general form of the clause is:

$$\left[\text{VALUE OF } \left\{\begin{matrix}\underline{\text{IDENTIFICATION}}\\ \underline{\text{ID}}\end{matrix}\right\} \text{ IS } \left\{\begin{matrix}\text{data-name-1}\\ \text{literal-1}\end{matrix}\right\} \left[, \underline{\text{DATE-WRITTEN}} \text{ IS } \begin{matrix}\text{data-name-2}\\ \text{literal-2}\end{matrix}\right] \left[, \underline{\text{LINES-PER-PAGE}} \text{ IS literal-3}\right] \left[, \underline{\text{LINES-AT-TOP}} \text{ IS literal-4}\right] \left[, \underline{\text{LINES-AT-BOTTOM}} \text{ IS literal-5}\right] \left[, \underline{\text{LINE-SPACING}} \text{ IS literal-6}\right]\right]$$

Unless label records are OMITTED, the VALUE OF ID clause should be used to furnish the standard label check routine with the proper identification for checking on input or writing on output.

The ID is a 30-character maximum Standard Data Format label which appears in the standard label record (to be defined at a later date).

These clauses may be written in any order, and any or all of them may be used for a given file, if pertinent.

When the DATE-WRITTEN option is to be used, literal-2 or the value of data-name-2 must be a positive numeric literal of the form:

MMDDYY

where MM represents month, DD the day, and YY the year.

The optional clauses LINES-PER-PAGE, LINES-AT-TOP, LINES-AT-BOTTOM, and LINE-SPACING are applicable only for peripheral output. The literals must all be positive numeric integers with values between 0 and 63.

If literal-6 is either 0 or 1, the format will be single spaced.

The ADVANCING option of the WRITE verb takes precedence over LINE-SPACING. (See WRITE, Section VII.)

If none of these four options is used, the format will be single spaced, with printing continuing right over the page seams, from one page to the next.

It should be noted that, even though these options are used, nothing will be printed except as a result of a WRITE statement. Furthermore, the contents of the line to be written are the responsibility of the COBOL programmer.

5. RECORD DESCRIPTION ENTRY (FILE SECTION)

Record Description entries are the means by which items of data are described to the COBOL compiler. Every item that is given a separate name must be described in a separate entry. Each entry defines the characteristics of a particular unit of data. With minor exceptions, each entry is capable of completely defining a unit of data. Because the COBOL detailed data descriptions involve a hierarchical structure, the contents of an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. (For explanation of subordinate entries, see discussion on Levels in this SECTION.)

The Record Description entry consists of a number of independent clauses. In the following discussion, the clauses that may be used in a Record Description entry will be covered separately. Rules for writing entries in the Common-Storage, Working-Storage, and Constant Sections will be presented later.

a. Entry Formats - General

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

b. Entry Formats - Specific

The individual clause formats are arranged here in alphabetic order, whereas the clauses in the "Complete Entry Skeleton" are shown in the recommended order.

c. Elements of a Detailed Data Description

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

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

1) A level number which shows the relationship between this and other units of data.

2) A data-name.

3) The SIZE in terms of the number of Standard Data Format characters.

4) The dominant USAGE of the data.

5) The number of consecutive occurrences (OCCURS) of elements in a table or list.

6) The CLASS or type of data; i.e., alphabetic, numeric or alphanumeric.

7) The indication of SIGNED.

8) Location of an actual or an assumed decimal point.

9) Location of editing symbols such as dollar signs and commas.

10) Justification and synchronization of the data. (JUSTIFIED, SYNCHRONIZED.)

11) Special editing requirements such as zero suppression and check protection.

12) Initial VALUE of a working-storage item or the fixed VALUE of a constant.

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

d. Complete Entry Skeleton

The function of the Record Description complete entry skeleton is to specify the characteristics of a particular item of data. The complete entry format follows:

Option 1

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

Option 2

level-number {FILLER / data-name} [REDEFINES...] [SIZE...]

[USAGE...] [OCCURS...] [SIGNED] [SYNCHRONIZED...]

[POINT...] [CLASS...] [PICTURE...] [JUSTIFIED...]

[editing clause...] [VALUE...]

Those clauses which begin with SIGNED, SYNCHRONIZED, POINT, PICTURE, and JUSTIFIED, as well as the editing clause, must be specified at the elementary item level.

The clauses may be written in any order with one exception: REDEFINES, when used, must immediately follow the data-name.

e. Class

The UNIVAC 1107 character set can be subdivided into numeric, alphabetic, and alphanumeric (alphameric) characters. Any data item can be classified as numeric, alphabetic, or alpha-numeric, depending on the characters it contains. The functions of the CLASS clause is to indicate to the compiler the class of the data as defined in this manner. The basic format is as follows:

$$\left[\text{CLASS IS} \left\{\begin{array}{l}\underline{\text{ALPHABETIC}}\\ \underline{\text{NUMERIC}}\\ \underline{\text{ALPHANUMERIC}}\\ \underline{\text{AN}}\end{array}\right\}\right]$$

AN is an acceptable abbreviation for ALPHANUMERIC.

These three classes may be described as follows:

Class	Description
NUMERIC	Consists entirely of digits (0-9); may also contain an operational plus or minus sign, or a blank in the sign position; an actual decimal point is a non-numeric character and is not permitted. If there is no sign associated with the data, the data is considered positive. If the data item is NUMERIC and no assumed decimal point is indicated, the datum is considered to be an integer.
ALPHABETIC	Describes data which contains any combination of the twenty-six (26) letters of the English alphabet, and the space. No other characters, and no numerals, can be used.
ALPHANUMERIC	Describes data which may contain any allowable character in the object computer's character set, including alphabetics and numerics. Thus, data which is ALPHABETIC or NUMERIC is also ALPHANUMERIC.

The CLASS clause may be written at any level. If a CLASS clause appears in an entry describing an item which is a group, it specifies the class for each item within the group; it must not be contradicted by the CLASS descriptions (if any) of the members of the group. It is not considered a contradiction for an ALPHANUMERIC group to contain an item which is either NUMERIC or ALPHABETIC, since the ALPHANUMERIC class contains the other two classes. Group items containing contradictory elementary items should not include a CLASS clause.

If an entry contains a PICTURE clause, a CLASS clause is not necessary. However, if both are used, they must be compatible.

Example of CLASS clause usage:

```
250500    01    ERRORS CLASS IS AN.                              7543004
```

f. Copy

The function of the COPY clause is to enable the programmer to include prewritten Record Description entries in the Data Division. These prewritten entries may be copied from elsewhere in the Data Division or from the COBOL library. Briefly, the effect of the COPY clause is to extract an entry or a series of entries from another place and put it into a program at the point where the COPY clause appears. The complete general form of a Record Description entry containing a COPY clause is the following:

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

The information being copied is inserted at the point in an entry where the COPY clause appears. Thus, level-number and data-name-1 are not replaced by the information being copied.

Copying ends when, in the entries being copied, a level-number is encountered which is numerically equal to, or less than, the level-number originally associated with data-name-2. In this way, one entry or many entries may be duplicated with one COPY.

The level-number of data-name-2 does not have to equal the level-number of data-name-1. If the level-numbers differ, all level-numbers of succeeding entries being copied will be adjusted appropriately by the difference between the level-numbers of data-name-1 and data-name-2.

When data-name-2 is an entry appearing elsewhere in the Data Division (rather than in the library), it may occur either before or after the entry referring to it. Data-name-2 must never be subscripted.

When information is to be copied from the library, the FROM LIBRARY option must be included. Then if data-name-2 is not the name of a level (01) entry level in the library, it must be qualified by the item containing it which does have a level-number of 1. This qualification is necessary, even though data-name-2 may be unique. An entry being copied may itself contain a COPY clause only if it refers to the library.

When the FROM LIBRARY option is not used, then the entry designated by data-name-2, and any entries subordinate to that entry, must not themselves contain COPY clauses.

Example:

```
232300     01     SHIPMENTS-NOT-COSTED  COPY DETAIL-SHIPMENTS.     7543004
```

g. Data-Name

The function of the DATA-NAME is to specify the name of the data being described, or to specify an unused portion of the logical record. The general form follows:

$$\left\{ \begin{array}{l} \text{data-name-1} \\ \underline{\text{FILLER}} \end{array} \right\}$$

FILLER may not have a VALUE clause, OCCURS clause, nor editing clauses. A PICTURE clause is disallowed.

Every Record Description entry must have a subject; that is, a name which is to be assigned to the item being described. This name must appear immediately after the level-number. Data-names can consist of a maximum of 30 characters, including hyphens.

The subject of an entry must never be in qualified or subscripted form. Even if data-name-1 is not unique, it must not be qualified at this point in an entry, because its level-number and placement in relation to other entries automatically supply qualification. Thus, it is correct to write:

```
05    VENDOR....
      06    NAME....
05    RETAILER...
      06    NAME....
```

The key word FILLER is used to name an item in a record. Under no circumstances may a FILLER item be referenced directly. (Reference may be made to a group item which has a FILLER entry contained within the group.) The SIZE and USAGE of the item must be specified. Since FILLER cannot be referenced directly, it may not be a group.

Example of data-name and FILLER:

```
250700    02  FILLER PICTURE IS X (30).                    7543004
250800    02  GRADE.                                       7543004
```

The 01 level to these two entries starts at line 250500 (sample problem in appendix).

h. Editing

The function of this clause is to permit suppression of non-significant zeros and commas, to permit floating dollar signs or check protection, and to permit the blanking of an item when its value is zero. This clause allows the programmer to specify the editing features without using the PICTURE clause.

The general form of the editing clause follows:

[{ZERO SUPPRESS / CHECK PROTECT / FLOAT DOLLAR SIGN} [LEAVING integer PLACES]] [BLANK WHEN ZERO]

These clauses can be applied only to elementary items and to items which edit NUMERIC values.

The rules for editing, as shown in the MOVE verb, specify that data items are moved in conformity with the Record Description format of the receiving item.

The three options, ZERO SUPPRESS, CHECK PROTECT, and FLOAT DOLLAR SIGN, all permit suppression of leading zeros. If the LEAVING option is not employed, suppression will stop as soon as either a non-zero digit or the decimal point (actual or assumed) is encountered. Specifically:

1) When ZERO SUPPRESS is specified, leading zeros or commas will be replaced by spaces. This statement will not replace zeros to the right of the decimal point.

2) When CHECK PROTECT is specified, leading zeros or commas will be replaced by asterisks.

3) When FLOAT DOLLAR SIGN is specified, the right-most character suppressed will be replaced by a dollar sign, and all other characters which are suppressed will be replaced by spaces.

If the LEAVING option is used, the suppression or insertion will stop at such a point as to "leave" the specified number of places to the left of the decimal point. For example, a five-digit number described by the clause CHECK PROTECT LEAVING 3 PLACES would edit the number 7 so that it would appear as **007 when printed. The integer must have an integral value.

The editing clause cannot be used to specify zero suppression to the right of a decimal point; the BLANK clause is used for that purpose. If a PICTURE clause is used in the same entry as an editing clause, the two must not be contradictory. When the BLANK WHEN ZERO option is used, the item will contain nothing but spaces if the value of the item is zero. Thus, all other editing requirements, such as ZERO SUPPRESS, CHECK PROTECT, etc., will be overridden.

More comprehensive editing features are available in the PICTURE clause. When any of the above options are used, the format of the item is assumed to contain editing symbols. Programmers should determine the value of each of these options in systems design.

Examples:

1) 02 ON-HAND SIZE IS 6 NUMERIC DIGITS ZERO SUPPRESS.
   02 ON-HAND SIZE IS 6 NUMERIC DIGITS CHECK PROTECT.
   02 ON-HAND SIZE IS 6 NUMERIC DIGITS FLOAT DOLLAR SIGN.

Examples:

PICTURE 9999V99

Before	After
000006	06 (Zero Suppress)
000006	****06 (Check Protect)
000006	$06 (Float Dollar Sign)

2) 02 AMOUNT SIZE IS 8 NUMERIC DIGITS ZERO SUPPRESS POINT LEFT 2 LEAVING 2 PLACES.

Before	After
00000006	00.06

3) 02 NET-BALANCE SIZE IS 8 NUMERIC DIGITS BLANK WHEN ZERO.

Before	After
00000000	Spaces

Note: No more than one of the editing clauses ZERO SUPPRESS, CHECK PROTECT, FLOAT DOLLAR SIGN is legal in a single record description entry.

i. Justified

The function of the JUSTIFIED clause is to specify non-standard positioning of a data item when less than the maximum number of characters may be present.

If the programmer wishes to specify right justification in lieu of left justification, or the reverse, he may do so by means of the JUSTIFIED clause, which has the following form:

$$\left[\underline{\text{JUSTIFIED}} \quad \left\{\begin{matrix}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{matrix}\right\}\right]$$

The JUSTIFIED clause cannot be used if a decimal point has been explicitly specified in this entry.

An item of data may be moved within the computer by means of a MOVE statement or as a result of computation or some other operation. If the location to which it is moved is larger in size than the data itself, it may be necessary to specify the position the data is to occupy in its new location. The standard rules for data positioning, within an item area, are:

1) Numeric data is aligned by decimal point with zero fill on either end as required.

2) When no decimal point is specified, numeric data is right justified with zero fill.

3) Alphabetic and alphanumeric data is left justified with space fill.

Please note that this clause is required only when standard rules of positioning are not desired.

j. Level-Number

The function of the level-number is to show the hierarchy of data within a logical record and to identify entries for condition-names, non-contiguous constants, and working-storage items. Every Record Description entry, as its first element, must begin with a level-number. The level-number is used to show the relation of the item to other items.

Level-numbers must be chosen from the integers 1 (01) through 49, 77, and 88. Level-number 1 (or 01) is applicable only to records, not to just a part of a record.

The level number 01 signals the first entry in each Record Description. This corresponds to the logical record on which the READ and WRITE verbs operate.

Special level numbers have been assigned to certain entries where there is no real concept of level:

1) Level-number 77 is used for non-contiguous items in constant and work areas.

2) Level-number 88 is used to denote condition-names. Entries having this level-number must follow immediately after the entry describing the data-name with which the condition-names are associated.

For a general discussion of level-number, see "Concept of Levels".

Refer to "DATA DIVISION SAMPLE ENTRY" for an example of level-numbers, including 77 and 88.

k. Occurs

The function of the OCCURS clause is to eliminate the need for separate entries for repeated data and to supply information required in the application of subscripts. The general form of the OCCURS clause follows:

[OCCURS integer-1 TIMES]

Integer-1 represents the exact number of occurrences and must be a positive numeric literal having an integral value greater than zero.

The OCCURS clause must not be used in a Record Description entry in the File Section having a level number of 01. This clause is not allowed in any level 77 or 88 entry.

The OCCURS clause is used in defining tables and other homogeneous sets of repeated data. When the OCCURS clause is used, the data-name which is the subject of this entry must be subscripted whenever it is used as an operand, regardless of the value of integer-1. Further, if this data-name is the name of a group item, then all data-names belonging to the group must be subscripted whenever they are used as operands. The data description clauses, associated with an item whose description includes an OCCURS clause, apply to each repetition of the item being described. For example:

```
01  TABLES CLASS IS NUMERIC.
    02  COST-CENTER-TABLE OCCURS 8 TIMES.
        03  COST-CENTER-COLUMN PICTURE IS S9(7)V99
            USAGE IS COMPUTATIONAL SYNCHRONIZED
            RIGHT OCCURS 70 TIMES.
```

Any reference to a particular entry of COST-CENTER-COLUMN in this table must have two subscripts provided; i.e., MOVE COST-CENTER-COLUMN (TABLE, COLUMN) TO COMMON-AREA.

This clause is required when the data might occur more than once. It causes storage assignment to be repeated the stated number of times, according to the description of the item. If the clause is not used, the number of occurrences is assumed to be one.

Refer to the listing in the appendix for an example of a two-level table. Please note especially lines 510500-510800 for the OCCURS clause use. Please note that the REDEFINES clause is necessary in this instance.

If within the scope of an OCCURS clause an item is synchronized, then the "increment" of the OCCURS is "modularized" so that each subscripted item will begin in the same character position.

If OCCURS clauses are "nested" more than 3 deep, a warning will be given.

1. Picture

1) General

The function of the PICTURE clause is to show a detailed picture of (1) an elementary item, (2) the general characteristics of the item and, (3) special reporting editing.

The clauses of a Record Description entry describe certain characteristics of the data so that the computer can properly process it. The PICTURE clause provides the compact way of specifying most of this information. Each character of a data item can be represented by a code chosen to describe its principal characteristics. The general form of the clause is as follows:

```
PICTURE IS any allowable combination of
          characters and symbols as des-
          cribed in this section. A maxi-
          mum of 30 characters is allowed. *
```

* The number of successive characters in a PICTURE must not exceed 30. For example, $$ZZ.99 is a PICTURE containing 7 characters, 9V99 contains 4 characters and 9(100) contains 6 characters. The number of characters in a PICTURE may be different than the number of character positions described by the PICTURE. For example, 9(100) equals 6 characters, but describes 100 numeric character positions. You cannot have, for example, 9(10000000000000000000000000000). This equals 31 characters.

When an integer is placed in parentheses following a PICTURE symbol (A, X, 9, P, Z, *, $, B, 0, -, or +) it indicates to the compiler the number of successive times that character is to be present. For example, P(2)9(5) is equivalent to PP99999 and will be so interpreted by the processor. Note that the indicated symbol must be followed immediately by a parenthesis without an intervening space.

A PICTURE can be used only at the elementary item level. (See SIZE in this Section for a relation to the SIZE clause.)

When this clause is used, no other clause is required in the same entry. However, other Record Description clauses may also be written, provided they do not contradict the PICTURE clause.

The PICTURE clause specifies the SIZE of an item, the CLASS, the operational sign and an assumed decimal point, if present.

Editing can be specified by the PICTURE clause. Editing may involve the suppression of certain characters and/or the insertion of others. For example, the digits representing a man's salary might be 0011545. They would be much more readable on a check in edited form, as $**115.45; moreover, the use of the asterisks would hamper an attempt to alter the amount. Editing data requires moving it to an area for which the proper editing symbols have been specified.

The allowable characters (or symbols) and their definitions are as follows:

Data Characters

A represents an ALPHABETIC character
X represents an ALPHANUMERIC character
9 represents a NUMERIC character

Operational Symbols Not Representing Character Positions

S equivalent to SIGNED (leftmost character only)
V assumed decimal point location
P assumed decimal point scaling position
H specifies that the field is to be represented in complement binary arithmetic; i.e., COMPUTATIONAL-1 usage.

Zero Suppression Characters Representing NUMERIC Characters to be Suppressed or Replaced When Zero

Z standard suppression (replacement by blanks or spaces)
* check protection (replacement by asterisk)

Insertion Characters

$ dollar sign (floating when more than one is written)
, comma
. actual decimal point
B blank or space
0 zero

Report Signs

The table below illustrates the manner in which an operational sign (plus or minus) will be interpreted and displayed when printed as the result of stipulating Report Signs in the PICTURE clause.

Report Sign	Operational Sign: Negative	Operational Sign: Positive		
-	-	blank		(floating when more than one is written)
+	-	+		
CR	CR	blanks	(2)	
DB	DB	blanks	(2)	

2) Picture Character Definition by Class

a) Numeric Items

A numeric item is an item which may contain only the numerals 0 through 9 and an operational sign. A numeric item may also have an assumed decimal point. The PICTURE of any numeric data item may contain only the following characters: 9, H, V, P and S.

Character	Meaning and Use
9	A 9 indicates that the character position will always contain a NUMERIC character. Numeric items used in arithmetic operations may contain only the symbols S, V, P, H, and 9 in their picture.
H	An H is an operational symbol used to inform the compiler that the item being described by this field is COMPUTATIONAL-1. The H is not counted in the item size. The use of H will not cause automatic synchronization. The normal rule for numerics will apply for justification.
V	A V indicates the position of an assumed decimal point. Since a numeric item can contain only numerals and an operational sign, the actual decimal point (the special character period) cannot be included. An assumed decimal point is used for decimal alignment of items involved in computation. An assumed decimal point does not occupy a character position at object time and is not counted in the size of an item. For example, if a data item is described as having a PICTURE of 99V99 and the digits 3725 are moved into it, the value would be treated in calculation as 37.25, but the size of the item would be four characters, not five. If it were printed, it would print as 3725 since the decimal point character is not actually present and is not counted in the size of the item.
S	The character S is equivalent in meaning and use to the SIGNED clause; it indicates the presence of an operational sign. When used, it must always be written as the leftmost character of the PICTURE. It is not counted in the size of an item.

P

The character P indicates an assumed decimal scaling position. It is used to specify the location of an assumed decimal point when the point is not within the number as it appears in source data. For example, an item whose value is 678 would be treated in an arithmetic procedure as though its value were 678000 if the PICTURE were 999PPPV or as though its value were .000678 if the PICTURE were VPPP999. The character P is never considered as part of the size of an item; in the above examples, the size would be three characters.

When P is used in a PICTURE in an entry in the Working-Storage Section or the Constant Section, and if the entry also includes a VALUE clause, then a zero must appear in the VALUE clause in each character position containing a P in the PICTURE clause.

The following examples show the PICTURE of an item with a VALUE clause in the same entry and the actual digits placed in storage as a result:

PICTURE	VALUE Clause	Actual Characters Placed In Associated Storage Area
999PPP	VALUE IS 146000	146
PPP99	VALUE IS .0003	30
VP9	VALUE IS .01	1
999PPP	VALUE IS 1234000	234
VPPP99	VALUE IS .000372	37

Note that the last two examples would each produce an error message when the object program is compiled and would cause truncation or zero fill according to the rules of the MOVE verb given in SECTION VII.

b) Alphabetic Items

As previously specified in the CLASS clause, an ALPHABETIC item can contain only the letters of the English alphabet (A through Z) and the space. The PICTURE of an ALPHABETIC item can contain only the characters A and/or B.

Character	Meaning and Use
A	The character A, when used in a PICTURE, indicates that the character position will always contain either a letter or a space. Each position represented by an A in the item picture is counted in the item size.

c) Alphanumeric Items

An ALPHANUMERIC item is an item which may contain any character in the UNIVAC III character set. Alphanumeric items are divided into two types: non-editing items and editing items.

(1) Non-Editing Items

The PICTURE of an ALPHANUMERIC non-edit item may contain only the characters 9, A, and X. The characters 9 and A have been discussed above.

Character	Meaning and Use
X	The character X indicates that the character position may contain any character in the UNIVAC 1107 character set. For example, the PICTURE AAAXXXX indicates that the described item has a size of seven characters, that the first three characters will always be alphabetic, and that the last four characters may be any characters.

(2) Editing Items

Data is edited by moving the data to an editing (receiving) item. Characters are automatically inserted and/or suppressed in conformance with the Record Description entry of the editing (receiving) item. Editing items can be used in conjunction with the MOVE verb. They can also be used with the GIVING option of arithmetic statements to edit numeric data for readability on external media.

Editing may be specified by use of PICTURE or by use of the editing clauses previously discussed. (See h. Editing, in RECORD DESCRIPTION ENTRY, SECTION VI.)

The PICTURE of an editing item may only contain:

9 V Z $ - + , . CR DB * 0 B

The use of 9 and V have been discussed under numeric items. V is the only one of the characters that does not occupy a character position.

If a numeric item containing the symbol H in its picture is moved to an editing area using 9's and other editing symbols, the characters of the source item will automatically be converted to Standard Data Format (6-bit) characters in the receiving area.

The "+" and the "-" signs may only appear at the beginning or the end of a PICTURE. The report signs CR and DB may only be used at the rightmost end of a PICTURE.

The balance of the editing characters are used to suppress and/or replace characters in accordance with the rules given in this section. Two general rules apply: (1) Except in the cases of 0 and B, suppression and/or replacement terminates on the character preceding the first non-zero digit or the decimal point, whichever is encountered first. Zeros following a non-zero digit (1-9) will not be suppressed or replaced. (2) If every data character position in a PICTURE reserved for source data contains suppression and/or replacement characters (other than 0), then all characters will be replaced by blanks if the value of the source data is zero. This rule is equivalent in effect to the BLANK clause.

(a) Zero Suppression Character

Character	Meaning and Use
Z	The character Z specifies zero suppression of the indicated characters. Zero suppress is the process of replacing left-most zeros by blanks.

The following table shows the results of zero suppression:

Source Item	Editing PICTURE	Edited Item
12345	ZZZ99	12345
00123	ZZZ99	123
00100	ZZZ99	100
00000	ZZZ99	00
00100	ZZZZZ	100
00000	ZZZZZ	

A Z must never be preceded by a 9, a B, or a 0.

A Z can be preceded in the PICTURE only by one of each of the following characters or symbols: $, +, -, a decimal point (V or "."); you can have all the commas (,) you like. A Z to the right of an indicated decimal point in a PICTURE may appear when only all numeric character positions are represented by Z's and represents the "blank when zero" option in PICTURE form; i.e., the item will be blanked when the value of the data is zero. All other editing requirements, such as check protection, insertion of commas or an actual decimal point, etc., will be overridden. It is a programmer's responsibility to guard against issuing blank checks.

(b) Insertion Characters

An insertion character is inserted into a report item, and does not replace any data. It appears in addition to the information moved to the item. The insertion characters are $, .+-CR 0 (zero) B DB. When any of these characters are used, the size of the editing item must be larger than the maximum number of digits which might be moved to the item.

The use of an Insertion character will cause that character to be placed in the corresponding position of an item. The characters "$", "," and "." can be used only with NUMERIC data. In this case, a maximum of 18 NUMERIC character

representations may be shown in a PICTURE. When the item is used as a source, these character positions are treated as containing data of the same CLASS as the symbol (e.g., the positions shown by the Insertion characters are treated as though an A, X, or 9 had been shown). Any item in excess of 18 NUMERIC characters must be handled by programming.

Character	Meaning and Use
$	The single dollar sign in the PICTURE specifies the dollar sign character for that position.

Source Item	Editing PICTURE	Edited Item
123	$999	$123
012	$999	$012
012	$ZZZ	$ 12
000	$ZZZ	
010	$ZZZ	$ 10

Note that the PICTURE of the item specifies four character positions including the $ and a maximum of three digits of data may be moved. An edited item will contain only one dollar sign.

An editing item can contain only one algebraic sign.

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

When the minus sign is used, 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 will appear if the item is positive; a minus sign will appear if the item is negative. Unsigned items are considered positive.

Source Item	Editing PICTURE	Edited Item
+33	-99	33
-33	99-	33-
33	-99	33
00	-99	00
+22	+99	+22
-22	+99	-22
20	99+	20+

0 — The character 0 specifies that a zero will be inserted at the position specified in the receiving edited item. The 0 will be counted in determining the edited item size. For example, if the digits 123456 are moved to an item with a PICTURE of 90090099099, the item would appear 10020034056. This provides a means for spreading an item for zero insertion. Zero insertion to the right of an assumed or actual decimal point is not allowed.

B — The character B performs the same function as 0 except blanks are inserted. For example, if the digits 123456 are moved to an item with a PICTURE of 9BB9BB99B99, the item would appear 1 2 34 56. This provides a means for spreading an item for alignment, readability, etc.

The comma will be inserted at the indicated position in the edited item. For example, the PICTURE 9,999 would cause 1107 to become 1,107. The comma will be suppressed if zero suppression has caused the elimination of all digits to the left of the comma.

This character represents an actual decimal point. When used to describe a character position, the data being edited is aligned by decimal point and an actual decimal (".") will appear in the indicated character position.

The integer 1107 would appear as $1,107.00 if the notation $9,999.99 were used as the editing PICTURE. The actual decimal point occupies a character position and is contained in the size of the editing item. A PICTURE must never contain more than one decimal point, assumed or actual. A PICTURE may not end with ".".

CR and DB

The credit and debit symbols CR and DB may appear only at the right end of a PICTURE. These symbols occupy two character positions. When the value of the described item is negative, the

specified symbol will be placed at the right end of the item. If the value of the item is positive, these character positions will contain blanks. For example, the PICTURE $99.99CR will cause -1107 to become $11.07CR and +1107 to become $11.07 after editing. The report signs CR and DB are always fixed.

(3) Replacement Characters

The Replacement characters used in PICTURE specify that certain digits will be replaced by these characters in the editing. Replacement characters are *, the floating dollar sign, the floating minus sign, and the floating plus sign.

Character	Meaning and Use
*	The asterisk is used to indicate check protection; i.e., the suppression of each specified zero on the left and its replacement by an asterisk. The following table illustrates the use of the asterisk.

Source Item	Editing PICTURE	Edited Item
11070	***99	11070
00110	***99	**110
00000	***99	***00
00000	*****	
00110	*****	**110

An asterisk can be preceded only by a dollar sign, a plus sign, a minus sign, a decimal point (V or .), and a comma.

Zero suppression with a floating dollar sign can be designated by using the symbol "$" to represent each leading numeric character into which the dollar sign may be floated. A single dollar sign will be placed in the least significant suppressed position shown by the symbol "$" in the PICTURE. This item must contain at least one more position than the maximum number of significant digits to be stored in it.

When zero suppression, check protection, floating dollar or floating report signs are designated in a PICTURE, suppression of leading commas is also provided.

Floating Report signs ("-" or "+") can be designated in the same manner as a floating dollar sign. The Report signs "CR" and "DB" are always fixed.

If a suppression character appears in each data position, the item will be blanked when the value of the data is zero. Thus, all other editing requirements, such as check protection, insertion of commas, or an actual decimal point, etc., will be overridden.

SUMMARY OF PICTURE SYMBOLS

NON EDITING

If PICTURE is:	And Field in memory is:	Then Field Will be used as:	And Class is
99999	56789	56789	NUMERIC
9(3)V9(2)	56789	567.89	NUMERIC
S999V99	(-)56789	(-)567.89	NUMERIC
XXXXX	ABCDE	ABCDE	ALPHANUMERIC
AAAAA	BCDEF	BCDEF	ALPHABETIC
X(5)	12345	12345	ALPHANUMERIC
XXXXX	AB67C	AB67C	ALPHANUMERIC
99X99	67.89	67.89	ALPHANUMERIC
99AA9	23FG4	23FG4	ALPHANUMERIC
A(2)X(3)	CD2M3	CD2M3	ALPHANUMERIC
99P(3)V	34	34000	NUMERIC
VP(3)99	34	.00034	NUMERIC
VP(3)9(2)	34	.00034	NUMERIC

EDITING

SOURCE AREA		RECEIVING AREA	
PICTURE	Field as in memory	PICTURE	Edited Field
9(5)	45678	$ZZ,ZZ9.99	$45,678.00
9(3)V99	45678	$ZZ,ZZ9.99	$ 456.78
9(3)V99	00067	$ZZ,ZZ9.99	$ 0.67
9(3)V99	00004	$ZZ,ZZZ.99	$ .04
9(5)	00000	$ZZ,ZZZ.ZZ	
V9(5)	12345	$ZZ,ZZ9.99	$ 0.12
H9(5)	12345	$**,**9.99	$12,345.00 (6-bit)
9(5)	00000	$**,***.**	
9(5)	67890	$$$,$$9.99	$67,890.00
H9(3)V99	67890	$$$,$$9.99	$678.90 (6-bit)
9(5)	00000	$$$,$$9.99	$0.00
9(5)	00000	$$$,$$$.ZZ	
V9(5)	67890 (6-bit)	$$$,$$9.99	$0.67 (6-bit)
SH9(5)V	(-)56789	-ZZZZ9.99	-56789.00 (6-bit)
SH9(5)	(+)56789	-ZZZZ9.99	56789.00
SH9(5)	(+)56789	+ZZZZ9.99	+56789.00
SH9(5)	(-)56789	+ZZZZ9.99	-56789.00
SH9(5)	(-)46789	$$$$$$.99CR	$46789.00CR
SH9(5)	(+)56789	$$$$$$.99CR	$56789.00

m. Point Location

The function of the POINT LOCATION clause is to specify the position of the assumed decimal point.

$$\left[\text{POINT LOCATION IS } \left\{\begin{matrix}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{matrix}\right\} \text{ integer PLACES}\right]$$

This clause is needed where a numeric item having non-integral values is not defined in a PICTURE clause.

This clause may only be used at the elementary item level.

POINT LOCATION must not be used if "." appears in the PICTURE clause.

If the PICTURE clause contains a V, the number of places specified in the POINT LOCATION clause must agree.

n. Redefines

The function of the REDEFINES clause is to allow the same computer storage area to contain different data items; i.e., to "overlap" 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 and cause both items to occupy the same physical area in storage. The general form of the REDEFINES clause follows:

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

The REDEFINES clause must immediately follow data-name-1. This is the only descriptive clause which must occur in a fixed place in an item description.

The following rules apply to the use of REDEFINES:

1) The level-numbers of data-name-1 and data-name-2 must be identical.

2) Redefinition starts at data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered.

3) When the level-number of data-name-2 is other than 1, it must specify a storage area of the same size as data-name-1.

4) The entries giving the new description of the storage area must immediately follow the entries describing the area being redefined.

5) This clause is not used at the record (01) level in the File Section; a DATA RECORD clause in the FD entry that names more than one data record implies automatic redefinition.

6) Data-name-2 should be qualified when necessary, but subscripting is not permitted.

7) The entries giving the new description of the storage area must not contain any VALUE clauses, except in condition-name entries.

When an area is redefined, all descriptions of the area remain in effect. Thus, if B and C are two separate items which share 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:

```
220700    03 QUANTITY.
220800       04 TONS PICTURE IS S9(4)V99 SYNCHRONIZED RIGHT.      7543004
220900       04 BASE-BOXES PICTURE IS S9(5) SYNCHRONIZED RIGHT.   7543004
221000       04 BASE-SYM REDEFINES BASE-BOXES.                    7543004
221100          05 FIRST-DIGIT PICTURE IS 9.                      7543004
221200          05 BASE-BOX-REST PICTURE IS S9(4).                7543004
```

In using the REDEFINES clause, the programmer must be extremely careful. If the areas utilized by each of the data-names are not equal, program errors are likely to occur.

To aid in determining area utilization (mapping), the following rules apply:

1) All 01 record description entries will begin in the leftmost Standard Data Character position of a computer word. If synchronization padding is required for the first entry, this padding will occur in the leftmost position for a synchronized right entry.

2) All independent entries (77 entries) will be noncontiguous to other 77 entries. Each entry will occupy one or more complete words according to the rules of justification for numeric and alphanumeric items.

When redefined areas are not equal, the next contiguous item will be assigned adjacent to the larger sized area, regardless of error warnings which may appear.

It is again emphasized that extreme care must be exercised, especially when either the SYNCHRONIZED clause, SIGNED, or the COMPUTATIONAL-1 option of the USAGE clause is used in either data-name entry.

If an item containing a SYNCHRONIZED clause also contains a REDEFINES clause, then the synchronization will occur with respect to the first full word at or beyond the beginning of data-name-2 in the REDEFINES clause. Should this result in inequities in area assignment, program errors are likely to occur.

o. Signed

The function of the SIGNED clause is to specify the operational sign of an elementary item. An operational sign is a reserved character or bit position that indicates to the computer whether a value is positive or negative. The general form of the clause is:

[SIGNED]

A signed item must be NUMERIC. Therefore, its CLASS need not be specified.

SIGNED indicates either a full 6-bit sign character, or a sign bit in complement arithmetic. (This may also be indicated by the use of an S in the PICTURE.) A standard operational sign is not considered in determining the SIZE. When an operational sign is specified in a PICTURE clause, the signed clause is not required.

The SIGNED clause cannot be used in an entry containing any of the editing options (ZERO SUPPRESS, CHECK PROTECT, FLOAT DOLLAR SIGN, and BLANK WHEN ZERO) as expressed in clauses or in PICTURE.

This clause can be used only at the elementary item level.

If an unsigned item is moved to a SIGNED item, the sign will be set to plus.

If a SIGNED item is moved to an unsigned item, the absolute value of the item is moved.

The sign of a decimal item (6-bit) occupies one character position preceding the first character position of the number.

p. Size

The function of the SIZE clause is to specify, in accordance with USAGE, the size of an item. The general form of this clause is as follows:

$$\left[\underline{\text{SIZE}}\ \text{IS}\ \text{integer-1}\ \left\{\begin{matrix}\text{CHARACTERS}\\ \text{DIGITS}\end{matrix}\right\}\right]$$

The size of an item must be specified at the elementary item level by means of a SIZE clause or a PICTURE. A PICTURE and a SIZE clause need not both be given. If both are given, they must agree. Use of the SIZE clause at any level other than the elementary item level is optional. The actual size of a group is the sum of the sizes of the elementary items comprising the group. Integer-1 represents the exact number of characters excluding operational signs.

Any of the key words of the USAGE and/or CLASS clauses can be inserted between integer-1 and the word CHARACTERS or DIGITS (whichever is used), in the SIZE format. If this is done, separate USAGE and/or CLASS clauses must not be written. For example:

SIZE IS 7 NUMERIC COMPUTATIONAL CHARACTERS

By the <u>area</u> of an <u>item</u> is meant the number of characters required in the computer by the item. The "size" of an item may differ from its area when it is SYNCHRONIZED, COMPUTATIONAL-1, or SIGNED. A warning will be given by the compiler if a group item is involved in a move and the areas of the sending and receiving items do not agree.

If the SIZE of a group item is specified and this SIZE disagrees with either the sum of the SIZEs of the elementary and/or with the area of the group, then a warning will be given.

The SIZE of an item may not exceed 262,143 characters.

q. Synchronized

The function of the SYNCHRONIZED clause is to specify positioning of an elementary item within a computer word or words. The general form of the SYNCHRONIZED clause is as follows:

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

This clause must be used at the elementary level only.

In the UNIVAC 1107 system, data is stored in computer "words" of fixed length. Thus, it is possible for the item SIZE to be smaller than the area allowed for it. This means that there will be unused "pieces" of words which are filled with non-significant characters, such as zeros or blanks. If the item is numeric, zero will be used; if the item is non-numeric, blanks will appear. However, it may be desirable to fill all available space with data, so that more than one item may be placed in one machine word. This is known as "packing".

If the SYNCHRONIZED clause is used, the compiler will understand that the item being described is not packed. That is, no other information occupies the machine word or words which contain this item. The clause is used to describe items in the Common-Storage, Working Storage and Constant Sections.

This clause indicates that the COBOL compiler, in creating the Internal Format of this item, must place the item in the minimum number of computer words which can contain the item.

If SYNCHRONIZED LEFT is specified, the leftmost character of the described item occupies the left-hand position of the next available machine word. If SYNCHRONIZED RIGHT is specified, the rightmost character in the item will occupy the right-hand end of a word.

Any unused character positions resulting from the SYNCHRONIZED clause may appear in the external media.

When SYNCHRONIZED is used in conjunction with SIGNED option, the sign of the item appears in the normal operational sign position of the computer, regardless of whether the item is SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT.

If synchronization is not specified, successive items of data are packed. Packing makes efficient use of storage space, but it may make each item relatively inaccessible and cause an increase in the running time of the object program. Therefore, if an item is referred to often, it may be advisable to synchronize it so that it can be obtained in less time.

See USAGE clause.

r. Usage

The function of the USAGE clause is to specify the dominant use and the mode of representation of a data item. Data is used in two principal ways: (1) in operations involving computation and (2) in operations in which the data is edited; i.e., "displayed" for reading. To improve the efficiency of the object program, it is desirable to indicate specifically or to imply to the COBOL compiler how an item will be used. The USAGE clause serves this purpose. The basic format is as follows:

$$\left[\text{USAGE IS } \left\{\begin{array}{l}\underline{\text{COMPUTATIONAL}}\\ \underline{\text{COMPUTATIONAL-1}}\\ \underline{\text{DISPLAY}}\end{array}\right\}\right]$$

The USAGE clause can be written at any level. The USAGE clause written at a group level applies to each elementary item in the group and the USAGE clause of an elementary item cannot contradict the USAGE of a group to which the item belongs.

If the USAGE clause is not specified for an elementary item, or for any group to which the item belongs, then the USAGE is assumed to be DISPLAY. DISPLAY indicates that the internal format of the data is a Standard Data Format.

The usage specified in this clause does not restrict in any way the operation of any verb on the data. However, it does affect the way in which the data is represented inside the computer, and this will affect the efficiency of the object program. Specifying one USAGE does not preclude an item from being used according to the other. For example, COMPUTATIONAL usage will be specified for most NUMERIC items; this does not prevent their being displayed. The opposite is also true.

If USAGE IS COMPUTATIONAL, the internal format of the data is Standard Data Format, that is 6-bit representation. The characters which may be used are restricted to 0 - 9 and + - or space in the sign position.

If USAGE IS COMPUTATIONAL-1, the format of the data is one's complement binary. If a COMPUTATIONAL-1 item is greater than 36 bits (10 characters: see table below), then the 36th bit is a sign bit, and the leftmost bit is a sign bit; i.e., the item will have two signs. The item must be SYNCHRONIZED RIGHT, and the SIZE must be expressed in CHARACTERS, according to the following table:

DECIMAL SIZE	BINARY BITS
1	6
2	12
3	12
4	18
5	18
6	24
7	30
8	30
9	36
10	36
11	42
12	42
13	48
14	54
15	54
16	60
17	60
18	66

Example:

```
06    KITTENS SIZE IS 10 CHARACTERS USAGE
          COMPUTATIONAL-1 SYNCHRONIZED RIGHT.
```

A COMPUTATIONAL or COMPUTATIONAL-1 item is capable of representing a value to be used in computations and must be NUMERIC. Therefore, the CLASS need not be given. If a group item is described as COMPUTATIONAL, the elementary items in the group are COMPUTATIONAL. The group item itself is not COMPUTATIONAL (i.e., cannot be used in computations).

The PICTURE of a COMPUTATIONAL and COMPUTATIONAL-1 item can contain only H, 9's, and operational symbols, S, V, and P.

s. VALUE

The function of the VALUE clause is to define the value of named constants, the initial value of working-storage items or the values associated with a condition-name. The general form of the VALUE clause is as follows:

Option 1 [VALUE IS literal]

Option 2 [{VALUE IS / VALUES ARE} literal-1 [THRU literal-2]]

Literal may be any numeric or non-numeric literal or figurative constant. The VALUE clause of the item and the clause governing CLASS for this item must not be contradictory. The VALUE clause may be used in an entry in the Working-Storage Section to specify the value of a condition-name or the initial value of an item. In the File Section, it may be used only to describe condition-names, and any other use of this caluse in that section is incorrect. The VALUE clause is used in the Constant Section to specify the value of a named constant.

Option 2 is used in a condition-name entry only and no further information is required.

1) The VALUE clause has no meaning in a file which is defined in the File Section, and if used, may cause a program error.

2) For Working-Storage or Common-Storage, the location must contain the specified VALUE at the start of the object program.

If the VALUE clause is used at the group level in the Working Storage or Constant Sections, it must not appear in the entries within the group. The group area will be initialized without consideration for the individual elementary, or other group items, contained within this group. Also, the VALUE clause must not appear in a Record Description entry containing an OCCURS clause or in an entry which is subordinate to an entry containing an OCCURS clause.

When VALUE is not specified, the initial contents in the Working-Storage areas may be unpredictable.

Examples:

The following is an example of the use of the VALUE clause to show the value of three condition-names associated with a data-name:

```
05  STUDENT-STATUS SIZE  1  COMPUTATIONAL.
    88  FRESHMAN VALUE IS  1.
    88  SOPHOMORE VALUE IS 2.
    88  JUNIOR VALUE IS 3.
    88  SENIOR VALUE IS 4.
```

The following examples might be used to specify the values of named constants or the initial values of items in work areas:

```
VALUE IS 1          (CLASS IS NUMERIC or ALPHANUMERIC.)
VALUE "ONE"         (CLASS IS ALPHABETIC or ALPHANUMERIC.)
VALUE IS ". 0"      (CLASS IS ALPHANUMERIC.)
VALUE IS SPACE      (CLASS IS ALPHABETIC or ALPHANUMERIC.)
VALUE "DICK"        (CLASS IS ALPHABETIC or ALPHANUMERIC.)
```

Refer to Data Division sample entry for an example of the VALUE clause used in a COBOL program.

t. Condition-Name

Each condition-name requires a separate entry with level number 88. This entry contains the name of the condition and the value associated with the condition-name. The condition-name entries for a particular conditional variable must follow the entry describing the item with which the condition-name is associated.

Only a VALUE clause may be used with a level-88 item.

More specifically:

```
nn  data-name.
    88 condition-name-1 VALUE IS literal-1.
    88 condition-name-2 VALUE IS literal-2 THRU literal-3.
```

As an example:

```
03  Grade  SIZE IS 2 CHARACTERS.
    88 FIRST VALUE IS 1.
    88 SECOND VALUE IS 2.
```

For use of condition-name, see CONDITIONS, Section VII.

Level 88 entries are allowed at the group level.

6. SPECIFICATION AND HANDLING OF LABELS

The UNIVAC 1107 COBOL compiler provides for the automatic handling of four types of labels--beginning and end, file and reel. This assumes the use of the UNIVAC 1107 Standard Tape Labels to be specified later.

The processing required to handle all other labels must be specified in the PROCEDURE DIVISION of the program.

7. COMMON-STORAGE SECTION

a. Concept of Common Storage

Common Storage is that part of computer memory set aside for intermediate processing of data and intercommunication between separately compiled sections of a COBOL program (see Section X). The difference between Common-storage and file storage is that the former deals with computer memory requirements for the storage of intermediate data results, whereas the latter deals with the characteristics of the entire file, as well as the computer memory requirements for the storage of each record of the file.

b. Organization

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

```
COMMON-STORAGE SECTION.

    77 data-name-1

        88 condition-name-1
           .
           .
           .
    77 data-name-n
           .
           .
           .
```

```
.
.
.
01 data-name-2

   02 data-name-3
      .
      .
      .
01 data-name-4

   02 data-name-5

   03 data-name-n

      88 condition-name-2
```

c. Non-Contiguous Common Storage

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

The following Record Description clauses are required in each entry:

1) Level-number

2) data-name

3) CLASS or PICTURE

4) SIZE (when PICTURE is not specified)

The OCCURS and REDEFINES clauses are not meaningful and will cause a diagnostic message at compilation time if used. Other Record Description clauses are optional and can be used to complete the description of the item, if necessary.

d. Common-Storage Records

Data elements in Common-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 COMMON-STORAGE Record Description, including REDEFINES, OCCURS, and COPY. Each working storage record-name (01 level) must be unique since it cannot be qualified by a file-name or section-name. Subordinate data-names need not be unique if they can be made unique by qualification.

e. Condition-Names

Any Common-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.

8. WORKING-STORAGE SECTION

a. General

The Working-Storage Section is used to describe areas of storage where intermediate results and other items are stored temporarily at object time. This section consists of a series of Record Description entries, each of which describes an item in a work area. The entries must be preceded by the header WORKING-STORAGE SECTION.

The difference between WORKING-STORAGE and FILE storage is that the former deals with computer memory requirements for the storage of intermediate data results whereas the latter deals with the characteristics of the entire file, as well as the computer memory requirements for the storage of each record of the file.

b. Organization

Whereas the FILE Section is composed of File Description entries and Record Description entries, the WORKING-STORAGE Section is composed only of Record Description entries. The WORKING-STORAGE Section begins with a section-header and a period, followed by Record Description entries for independent work area items, and then by Record Description entries for group work area records, in that order. The skeletal format for the WORKING-STORAGE Section is as follows:

```
WORKING-STORAGE SECTION.
77      data-name-1.
        88  condition-name-1.
                .
                .
77      data-name-n.
01      data-name-2.
        02  data-name-3.
            .
            .
            .
```

c. Non-contiguous Working Storage

Non-contiguous working storage consists of a single item which is not subdivided and is not itself a subdivision of some other item. It is always assigned the level-number 77. Each non-contiguous item must be described in a separate Record Description entry consisting of the following parts:

The level-number 77.
A data-name.
A CLASS or PICTURE clause.
A SIZE clause (if a PICTURE clause is not used).

The OCCURS clause must not be used in describing a non-contiguous item. The OCCURS clause is not meaningful and will cause an error at compilation time if used. Other Record Description clauses are optional and can be used to complete the description of the item, if necessary. When writing the Working-Storage Section, entries for all non-contiguous items are placed before the entries describing grouped items.

Non-contiguous working storage is 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, but that 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 non-contiguous working storage.

d. Group Work 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, including REDEFINES, OCCURS, and COPY. Each working-storage record-name (01 level) must be unique since it cannot be qualified by a file-name or section-name. Subordinate data-names need not be unique if they can be made unique by qualification.

e. Initial Values

The initial value of any item in the WORKING-STORAGE section may be specified by using the VALUE clause of the Record Description. If VALUE is not specified, the initial value may be unpredictable. All the rules for the expression of literals and figurative constants apply. The size of a literal used to specify an initial value can be equal to or less than the size specified in the SIZE clause (or PICTURE) of the associated data entry, but it cannot be greater. When the size is less, normal rules for a MOVE of a literal apply.

For example:

1) 77 PAGE SIZE IS 7 NUMERIC COMPUTATIONAL CHARACTERS VALUE IS 0000342. (Legal)

2) 04 PAGE-NO SIZE 4 NUMERIC DIGITS VALUE IS 5. (Legal)

3) 02 ADDRESS SIZE 5 CLASS AN CHARACTERS VALUE IS "XY1245Z". (Illegal)

4) 03 GROUPING SIZE 9 NUMERIC DIGITS VALUE IS ZEROS. (Legal)

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

f. Condition-Names

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

Refer to DATA DIVISION SAMPLE ENTRY for an example of a Working-Storage Section used in a COBOL program.

9. CONSTANT SECTION

a. General

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

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

Constant storage is, therefore, that part of computer memory which is set aside to save named constants for use in a given program; a named constant being a named item having a value that does not change during the course of a program. The Constant Section of the Data Division contains the entries which describe named constants and specify their values. Items in the Constant Section can not be the result of an operation.

b. Organization

The Constant Section is organized in exactly the same way as the Working-Storage Section, beginning with the header CONSTANT SECTION and contains only Record Description entries. The entries are of two types, non-contiguous and grouped. The skeleton format for the CONSTANT SECTION is as follows:

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

c. Non-contiguous Constants

An independent non-contiguous constant describes a single item which is not subdivided and is not itself a subdivision of some other item. It is always assigned the level-number 77. Each non-contiguous item must be described in a separate Record Description entry consisting of the following parts:

The level-numer 77.

A data-name.

A CLASS or PICTURE clause.

A SIZE clause (if a PICTURE clause is not used).

A VALUE clause.

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

d. Grouped Constants

Named constants in CONSTANT SECTION which bear a definite relationship to one another must be grouped into records according to the rules for formation of Record Descriptions.

Record Descriptions can be used in a Constant Record Description, including REDEFINES, OCCURS, and COPY. Each Constant Section record-name (01 level) must be unique since it cannot be qualified by a file-name or section-name. Subordinate data-names need not be unique if they can be made unique by qualification.

e. Value of Constants

In the definition of constants, the VALUE clause is required and can only be specified for data items containing homogeneous characters; i.e., characters having the same USAGE clause. VALUE cannot be specified at levels which contain characters having different USAGEs nor can VALUE contradict the CLASS of the item.

All the rules for the expression of literals and figurative constants apply. The size of a literal used to specify the value of a constant can be equal to or less than the size specified in the SIZE clause of the associated data entry, but it cannot be greater. If the size of the literal is less, standard rules for justification apply.

f. Condition-Names

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

g. Tables of Constants

Reference data is often organized in the form of a "table". The COBOL system provides two methods of constructing such tables. One method involves naming and describing each item of data individually. The second method involves the use of subscripting.

If the programmer wishes to employ subscripting to obtain items from a table, he must first describe the table in the Constant Section as a group of constants. He must then redefine the table by means of a REDEFINES clause, accompanied by an OCCURS clause which furnishes the compiler with the information necessary to permit subscripting. See Section 4 for a general discussion of subscripting.

VII. PROCEDURE DIVISION

1. GENERAL DESCRIPTION

The PROCEDURE DIVISION contains those procedures needed to solve a given problem. (This is commonly called the program, but is really only one of the four parts of the program). These procedures are written as sentences, combined to form PARAGRAPHS, which in turn may be combined to form SECTIONS.

2. RULES OF PROCEDURE FORMATION

a. General

COBOL procedures are expressed in a manner similar (but not identical) to normal English prose. In the English language verbs designate action. So it is with the COBOL verbs, which form the basis of the Procedure Division. The verbs specify the operations (or procedure) to be executed to solve a given problem. The basic unit of procedure formation is the sentence. One or more sentences form a paragraph while a paragraph or a group of paragraphs form a section. The Procedure Division is comprised of a paragraph, a group of paragraphs, a section or group of sections.

Most verbs will cause some action to be taken at the time the program is executed. The verb DISPLAY will cause specified information to be typed or otherwise displayed, and the verb 'STOP literal' will cause the computer to halt.

b. Format

All paragraph names must start at position A (column 8 of the Reference Format), be thirty characters or less in length and be followed by a period.

Each verb in COBOL has fixed formats in which it can be used. The format specifies the arrangement of a verb and its operands. This arrangement defines a particular type of procedure statement. The specific format for each verb is shown in this section.

c. Statements

There are three types of statements (also called sentences): imperative statements, conditional statements, and compiler directing statements.

1) Imperative Statements

An imperative statement consists of either a verb (excluding compiler directing verbs) and its operands, or a sequence of imperative statements. A sequence of imperative statements may contain either a GO imperative statement or a STOP RUN imperative statement, which (if present) must appear as the last imperative statement of its (GO or STOP RUN) sequence. If a GO TO verb is not present, an imperative statement is executed in its entirety and control is passed to the next procedural statement.

2) Conditional Statements

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

(1) IF condition { statement-1 / NEXT SENTENCE } [{ OTHERWISE / ELSE } { statement-2 / NEXT SENTENCE }]

or

(2) Imperative statement followed by a conditional statement.

In form (2) the imperative statement may be either an arithmetic verb or a READ verb and its operands. The conditional statement may be the ON SIZE ERROR used with the arithmetic verbs, or the AT END option used with the READ verb.

Statement-1 cannot be conditional; statement-2 can be either imperative or conditional, and if conditional, can, in turn, contain no additional conditional statement. The condition within the conditional statement is considered to be "nested".

The key words NEXT SENTENCE may be substituted for either statement-1 or statement-2.

The truth or falsity of a condition is determined and if the condition is found to be true, control is passed to statement-1 of the conditional sentence. If the condition is found to be false, control is transferred to the next sentence, or to the statement following OTHERWISE/ELSE if either of these key words has been employed.

Examples of Conditional Statements:

(1) Simple Condition

```
630300  IF DESC-CODE IN A-FIELD IS LESS THAN 963 GO TO DONT-CALC      7543004
630400  ELSE GO TO CK-NO-COST.                                        7543004
```

(2) Compound Condition

```
602400  IF WORKS IN A-FIELD IS EQUAL TO 10 OR 15 OR 16 OR 18 OR 20   7543004
610100  OR 21 OR 27 OR 29 GO TO DONT-CALC                            7543004
```

(3) Compound Condition

```
650700  IF TONS IN B-FIELD IS EQUAL TO ZEROS AND BASE-BOXES IN       7543004
650800  B-FIELD IS EQUAL TO ZEROS GO TO WR-OUTPUT                     7543004
```

Note in example 2 that if WORKS is equal to <u>any</u> of the objects, control will be passed to DONT-CALC, whereas in example 3 <u>both</u> TONS and BASE-BOXES must be equal to zero before control is passed to WR-OUTPUT. This is true because of the use of the logical operator OR in example 2 connecting the various items to be compared while AND was used connecting the conditions in example 3.

[{ READ file-name-1 AT END / arithmetic-statement-2 on SIZE ERROR } imperative-statement]

The above represents a special form of conditional statement. It can be used only with the arithmetic verbs and the verb READ and its form is incorporated in the basic formats prescribed for those verbs. It differs from other conditional statements because of the nature of the conditional expressions which are evaluated. There are two of these, as follows:

(1) The AT END condition. When the records in a file of data are being made available to the object program by means of READ statements, the programmer may wish a particular action to be carried out if the end of the file is reached. The words AT END constitute an alteration of the sequence of a program when an end of file is found. The words AT END or, simply, the word END) must be followed by imperative-statement which prescribes the action to be carried out if the condition is found to exist.

(2) The SIZE ERROR condition. A computational result may be larger than the space the programmer has allowed for it in storage, in this case, a "size error" occurs, and it may have unforeseen effects. The programmer can specify a test to determine if a size error has occurred. The special condition used for this consists of the words ON SIZE ERROR. In the format shown, arithmetic-statement represents a statement employing one of the arithmetic verbs. It is followed by the words ON SIZE ERROR and then imperative-statement, which prescribes the action to be followed if a size error is found.

Examples:

```
READ INVENTORY-RECORD AT END PERFORM CLOSE-OUT-ROUTINE.
ADD MISCELLANEOUS TO TAX GIVING DEDUCTIONS ON SIZE ERROR
GO TO ERROR-CALC ROUTINE.
```

3) Compiler Directing Statements

These statements are instructions directed to the compiler; they cause the compiler to take certain specific action at compilation time.

d. Sentence Formats

The following rules apply to the punctuation of sentences:

(1) A sentence is terminated by a period.
(2) A separator is a word or character used for the purpose of enhancing readability. Use of a separator is optional.
(3) The allowable separators are:
    ;
    THEN
(4) These separators must not be followed immediately by another such separator.
(5) Separators may be used in the following places:
    (a) Between statements.
    (b) In a conditional statement:
        (1) Between the condition and statement-1.
        (2) Between statement-1 and OTHERWISE.

e. Paragraphs

COBOL sentences may be combined to form paragraphs. A paragraph may contain one or more sentences. Every paragraph must begin with a procedure-name (i.e., there must be no unnamed paragraphs). A sentence within a paragraph cannot be assigned a procedure-name, but a paragraph may consist of only one sentence.

In writing procedures in accordance with the rules of the PROCEDURE DIVISION and the requirements of the REFERENCE FORMAT, the source programmer begins a paragraph with a name. The smallest grouping of the PROCEDURE DIVISION which can be named is a paragraph.

The source programmer will usually place compiler directing sentences in their own paragraphs. Paragraphs comprised of compiler directing sentences are called "Compiler Directing Paragraphs". Paragraphs which contain at least one procedural sentence are called procedural paragraphs.

f. 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. Sections must always be named. 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 at all if the programmer does not find it convenient.

A section consists of one or more successive paragraphs and when designated must be named. The section-name is followed by the word SECTION, a priority number which is optional, and a period. The section-name applies to all paragraphs following it until another section-name is found. It is not required that a program be broken into sections.

For use of SECTION to provide overlay of program segments see Special Features, Section X.

3. SENTENCE EXECUTION

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

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

To review Statements (sentences):

a. IF condition { statement-1 / NEXT SENTENCE } [{ OTHERWISE / ELSE } { statement-2 / NEXT SENTENCE }]

In the conditional sentence above, the condition is an expression (see VII. 3. 2) which is true or false. If the condition is true, then statement-1 is executed and control is transferred to the next sentence. If the condition is false, statement-2 is executed and then control is passed to the next sentence.

Compiler Directing Sentences each consist of one compiler directing statement and direct the COBOL compiler to take action at compilation time. On the other hand Procedural Sentences denote action to be taken by the object program.

4. RELATIONS

a. Relations

The need frequently arises in programming to test conditions in order to determine what action should be taken. COBOL provides a number of relations for the programmer to use to express the conditions he wants to test. For example, the statement IF BALANCE EQUAL TO ZERO is built around the relationship specified by the word EQUAL, which is a relation.

A list of the relations in the COBOL language follows:

IS [NOT] GREATER THAN
IS [NOT] EQUAL TO
IS [NOT] LESS THAN

Words in the foregoing list which are underlined must be present when the relation is used. Words which are not underlined may be omitted with no resulting effect on the meaning of the relation.

Relations are combined with data-names and literals, etc. to form conditions. The detailed rules for using relations will be presented later in this section under Relation Tests.

b. Logical Connectives

The two logical connectives are AND and OR. For example, IF AGE IS LESS THAN MAX-AGE AND AGE IS GREATER THAN UNDER-AGE MOVE NAME TO LIST. In this example both conditions must be true before the MOVE will occur.

c. Implied Subjects and Relations

In some cases, relations and subjects may be implied in a series of consecutive simple conditions. Thus, the expression AGE EQUAL TO 19 OR AGE EQUAL TO 28 OR AGE EQUAL TO 31 could be written as AGE EQUAL TO 19 OR 28 OR 31. Not only is the subject, AGE, implied in the last two conditions, but the relation, EQUAL TO, is also implied.

The following rules apply to the use of implied subjects and objects:

1) A relation can be implied only in a simple condition where the subject is also implied.

2) When a relation is implied, it is assumed to be that of the nearest completely stated simple condition to the left.

5. CONDITIONS

a. General

Conditional procedures are keystones of data processing problems. The purpose of a conditional statement is to cause the object program to select between alternate paths depending upon the truth or falsity of the condition. COBOL makes available to the programmer several means of expressing conditional procedures.

A condition is an expression which, taken as a whole, may be either true or false, depending on the circumstances existing when the condition is evaluated.

In COBOL, conditionals generally involve the key word IF followed by the condition to be evaluated followed by the procedures to be executed. Depending upon the truth or the falsity of the condition, different sets of procedures will be executed. COBOL conditions are of two types: simple and compound;

b. Simple Conditions

1) General

The basic type of condition is the simple condition. The general form in which a simple condition is written is:

$$\left\{\begin{array}{l}\text{data-name}\\\text{literal}\end{array}\right\} \text{ IS } \left\{\begin{array}{ll}[\text{NOT}] & \text{GREATER THAN}\\[\text{NOT}] & \text{LESS THAN}\\[\text{NOT}] & \text{EQUAL TO}\end{array}\right\} \left\{\begin{array}{l}\text{data-name}\\\text{literal}\end{array}\right\}$$

The data-name or literal to the left of the relation is called the subject of the simple condition. The data-name or literal on the right of the relation is called the object. The subject and the object of any given simple condition must not both be literals.

The terms GREATER THAN, LESS THAN, and EQUAL TO may be replaced by the symbols >, <, and =, respectively, wherever they appear throughout the Guide.

Examples:

U1107 IS GREATER THAN ANY-OTHER-MACHINE

AGE > 21

Note that the word NOT is used to specify the negation of the relation appearing immediately to its right. For example, AGE NOT > 21 is the negation of the condition, AGE > 21.

c. Alternate Forms of Simple Conditions

These forms can be used only with numeric data. The general form is:

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

The specific interpretations of these terms are: an item is POSITIVE only if its value is greater than zero. An item whose value is zero is NOT POSITIVE. An item is NEGATIVE only if its value is less than zero. An item whose value is zero is NOT NEGATIVE. More briefly, the value zero is considered neither positive nor negative.

6. DETERMINATION OF TRUTH OR FALSITY OF A SIMPLE CONDITION

a. Relation Tests

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

The basic means the computer uses in comparing any two characters is its 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. Therefore, generally it is meaningful to compare any character to any other character. The result will depend on the relative position of each character in the machine's collating sequence. The collating sequence for the UNIVAC 1107 is as follows (ascending order):

ABCDEFGHIJKLMNOPQRSTUVWXYZ)-+<=>$*(:, 0123456789'/.

(1) Comparison of NUMERIC Items

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

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

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.

(2) Comparison of non-NUMERIC Items

For two non-NUMERIC items, or one NUMERIC and one non-NUMERIC item, a comparison results in the determination that one of the items is LESS THAN, EQUAL TO, or GREATER THAN the other with respect to the ordered character set of the UNIVAC 1107 If a SIGNED, COMPUTATIONAL item is compared with a non-numeric item the sign will be ignored. There are two cases to consider: equal length items, and unequal length items. In a comparison of two non-numeric items, each character in an item is compared with the corresponding character of the other item. The comparison begins with the leftmost character of each item. If these two characters are found to be 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 end of the item is reached, whichever comes first. The items are determined to be EQUAL when the low order end is reached.

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

(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, in which case the two items are EQUAL.

For example, if the items REB1DP5 and REB2DP1 were compared, then REB2DP1 would be greater than REB1DP5 since 2 is greater than 1. If REB1 were compared with REB1DP2 , then REB1DP2 would be GREATER THAN REB1 since the characters of REB1 would be exhausted without detection of a difference and the last three characters of REB1DP2 are not spaces. If REB2 were compared to REB1DP2CLMI the first item would be greater because the comparison would end on the fourth character from the left in both fields, where 2 is greater than 1.

b. Condition-Names

A condition-name is a name given to one value of a data-name. In the Data Division, a condition-name is assigned to a particular value of a particular data-name. For example, in a program processing a payroll, the data item SHIFT-STATUS might be a code indicating whether an employee is on the day, evening, or midnight shift. If SHIFT-STATUS has the value 1, the employee is on day shift, if it has the value 2, he is on evening shift, and if it equals 3, he is on midnight shift. To determine whether or not an employee gets shift pay, the programmer could test this condition by using a simple relational condition in a conditional statement such as IF SHIFT-STATUS EQUAL TO 2 ADD 2ND-SHIFT-DIFFERENTIAL TO PAY-RATE.

However, if he so chooses, he can associate a condition-name with any or all of the values that SHIFT-STATUS might assume. Thus, in the Data Division, the condition-name DAY-SHIFT might be associated with the "condition" that the data-name SHIFT-STATUS has a value of 1. SECOND-SHIFT might be similarly associated with 2, and MIDNIGHT with 3. Then, as a "shorthand" form of the simple relational condition SHIFT-STATUS EQUAL TO 2, the programmer could write the single condition-name SECOND-SHIFT. Therefore, the following two statements would produce identical results.

```
IF SHIFT-STATUS EQUAL TO 2 ADD 2ND-SHIFT-DIFFERENTIAL TO PAY-RATE.
IF SECOND-SHIFT ADD 2ND-SHIFT-DIFFERENTIAL TO PAY-RATE.
```

The condition-name, then, is another form of a condition used as alternative way of expressing conditions which could be expressed by a simple relational condition.

NOTE: The construct "NOT condition-name" is allowed.

c. Compound Conditions

A compound condition is a sequence of simple conditions in which one of the words AND or OR appears between the conditions. The words AND and OR are called logical connectives when used in this sense.

The most general form of a compound condition is:

Compound-condition =

{ simple-condition-1 AND simple-condition-2 AND...AND simple-condition-n
{ simple-condition-1 OR simple-condition-2 OR...OR simple-condition-n

Suppose the following conditions were being used in a program:

MARRIED
HOURLY-RATE IS LESS THAN 5.00
HOURLY-RATE IS GREATER THAN 3.50

These conditions could be linked by AND or OR in any sequence which would express a desired "overall" condition.

MARRIED AND HOURLY-RATE LESS THAN 5.00 AND
HOURLY-RATE GREATER THAN 3.50

1) Rules For Forming Compound Conditions:

(1) Two or more simple conditions connected by either ANDs or ORs make up a compound condition.

(2) The word OR is used to mean "either or both". Thus, the expression A OR B is true if: A is true, or B is true, or both A and B are true.

(3) The word AND is used to mean "both". Thus, the expression A AND B is true if and only if both A and B are true.

(4) The words AND and OR cannot both appear as logical connectives within the same compound condition.

(5) If a compound condition consists of several consecutive simple conditions, and if all of these conditions have the same subject and the same relation then the common elements may be implied instead of explicitly repeated in each condition as previously discussed under Implied Subjects and Objects.

The following examples of compound conditions may help to clarify the above rules:

Compound Condition	Interpretation
MARRIED OR DIVORCED	Either or both of these two conditions must be true for the expression as a whole to be true.
MARRIED AND AGE NOT GREATER THAN 21	The condition MARRIED must be fulfilled and the condition AGE GREATER THAN 21 must not be fulfilled for this expression to be true.

d. Class Tests

The characters in the UNIVAC 1107's character set are divided into three classes which are defined as follows:

1. The NUMERIC class includes the digits 0 through 9 and the operational signs, + and - .

2. The ALPHABETIC class includes the letters of the alphabet and the space.

3. The ALPHANUMERIC class includes every character in the UNIVAC III's character set.

The Class test determines at object time whether the item contains data which is either wholly numeric or wholly alphabetic. The form in which the class test is written is:

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

The data-name must be defined in the Data Division as being of the ALPHANUMERIC class.

7. VERBS

a. General

COBOL verbs form the basis for the Procedure Division. The verb specifies the specific procedures to be followed to accomplish a given problem solution.

Each verb has one or more fixed formats for use. The format indicates the arrangement of the verb and its operands and defines a particular category of procedure statements. The categories function and verbs follow:

Category	Verb
Arithmetic	ADD SUBTRACT MULTIPLY DIVIDE
Input-Output	READ WRITE OPEN CLOSE ACCEPT DISPLAY MONITOR
Procedure Branching	GO ALTER PERFORM
Data Movement	EXAMINE MOVE
Ending	STOP
Compiler Directing	ENTER COPY NOTE EXIT RETURN

b. Specific Verb Formats

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

(1) ACCEPT

The function of the ACCEPT verb is to receive low volume data from an available hardware device. The format is:

ACCEPT data-name [FROM mnemonic-name]

Mnemonic-name corresponds to a hardware device, either the console typewriter or the paper-tape reader as defined under SPECIAL-NAMES in the ENVIRONMENT DIVISION.

If the format of data-name contains fewer than the maximum number of characters, the data will appear in the leftmost positions of the input area associated with the device. The compiler will provide appropriate instructions in the object program to accommodate any difference in size and to move the data-name area.

Example:

```
601000    ACCEPT PRESENT-DATE                7543004
```

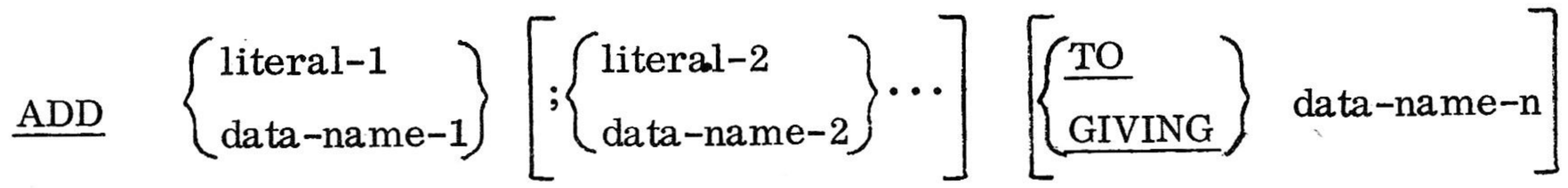
Data accepted from the console will be placed at PRESENT-DATE.

The object program will not continue until the ACCEPT is satisfied.

When the ACCEPT FROM option is used the mnemonic-name assigned to one of the hardware-names listed in the Environment Section, Section VIII, will be used.

(2) ADD

The function of the ADD verb is to add two or more numerical data items and set the value of an item equal to the result. The format is:



An ADD statement must name at least two addends. Thus the minimum ADD statement is of the form:

ADD data-name-1 data-name-2.

or

ADD data-name-1 TO data-name-n.

When the GIVING option is used, the sum of the values of the preceding data-names and/or literals will be placed in data-name-n. Since data-name-n, in this instance, is a receiving item, it may be an editing item.

When the TO option is used, the values of the data-names, including data-name-n, and literals in the statement are added and the resulting sum becomes the value of data-name-n.

If neither the GIVING nor the TO options are used, the last named (i.e., the rightmost or last written) addend must not be a literal. The sum of the values of all the data-names and/or literals will be calculated. The value of the right-most data-name will then be made equal to the sum.

The following rules apply to the ADD verb:

a. All data-names used in arithmetic statements must represent numeric data items that are defined as elementary items in the Data Division of the program. An item in Constant Storage cannot be a receiving item and must not appear in an arithmetic statement as the item receiving the result.

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

c. The maximum size of any addend (data-name or literal) is 18 decimal digits. This does not apply to intermediate results which will be carried out to 21 digits. These intermediate results are carried out to one more digit on the right and to at least two more on the left than is specified for the final results.

d. The formats of the two or more operands in an arithmetic statement may differ from each other; e.g., the programmer can add data-name-1 to data-name-2 even though the PICTURES of the two operands are 99V9 and H9V99, respectively. Decimal point alignment is supplied automatically throughout computations. Conversion of items with unlike usage will also be automatic.

e. The format of any data item involved in computations (e.g., addends, subtrahends, multipliers, etc.) cannot contain editing symbols. If this rule is violated, the processor will indicate the error by an appropriate message. Operational signs and implied decimal points are not considered editing symbols. The data-name in the GIVING option represents data items which must not enter into computations if they contain editing symbols.

f. The only figurative constant permitted in arithmetic statements is ZERO (including ZEROS and ZEROES).

If the number of decimal places in a computed result (sum, difference, product or quotient) exceeds the number of decimal places in the format of the data-name associated with the result (i.e., the data-name that is to take on the value of the result), truncation will occur unless the ROUNDED option has been used.

Truncation is the dropping of excess digits; it is always determined by the format of the data-name 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.

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 data-name, a size error condition arises. In the event of a size error condition, one of two possibilities will occur, depending on whether or not the ON SIZE ERROR option has been specified.

a. 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 effect is unpredictable.

b. If the ON SIZE ERROR option has been specified, and a size error condition arises, then the value of the resultant data-name will not be altered. Instead, the "any imperative statement" associated with the ON SIZE ERROR option will be executed.

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

Examples:

```
611500      ADD 1 TO INCREMENTOR.                                    7543004
```

When neither the TO nor the GIVING option is used, the last named addend cannot be a literal.

```
632200      MOVE ZEROS TO USS-FOB-COST IN C-FIELD.                   7543004
632300      ADD USS-INV-COST IN B-FIELD TO USS-FOB-COST IN C-FIELD    7543004
632400      ROUNDED                                                  7543004
```

USS-INV-COST IN B-FIELD has a picture of SH9(7)V99 while USS-FOB-COST IN C-FIELD has a picture of SH9(9). The object of the instructions shown is to place the whole dollars of USS-INV-COST IN B-FIELD into the area described as USS-FOB-COST IN C-FIELD. Assume that USS-INV-COST IN B-FIELD contains the amount 0004734V75. If this amount is moved to USS-FOB-COST IN C-FIELD, truncation will occur and the receiving area will contain 000004734. By filling this area with zeros, however, and utilizing the ADD verb with the ROUNDED option, the area will then contain 000004735.

(3) ALTER

The function of the ALTER verb is to modify the effect of GO TO sentences elsewhere in the program and thus to change a predetermined sequence of operations. The format is:

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

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

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

Examples:

```
670110  EX-HDR.                                                          7543004
670120      GO TO SET-MON.                                               7543004
670130  SET-MON.                                                         7543004
.
.
.
671100  EX-HDR-A                                                         7543004
671200      GO TO EX-HDR-REST.                                           7543004
671300  EX-HDR-REST.                                                     7543004
671400      ALTER EX-HDR-A TO PROCEED TO SET-UP-EX  EX-HDR               7543004
671500      TO PROCEED TO EX-HDR-MV.                                     7543004
```

The two GO TO paragraphs are 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.

A reference by an ALTER to a paragraph which consists of a GO TO statement and a NOTE statement will not be recognized as an error. Control will be transferred to the next statement in sequence, following the execution of the ALTER statement.

(4) CLOSE

The function of the CLOSE verb is to terminate the processing of one or more input or one or more output files or reels and to provide for optional rewinding and/or locking. The format is:

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

File-name-1 refers to the FD level name and must have an OPEN statement executed prior to the CLOSE statement.

CLOSE file-name

The CLOSE file-name option (as applied to the entire file rather than to individual reels) will initiate the final closing conventions for the file and release 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:

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

b) If the NO REWIND option is specified, the current reel of the file will remain in whatever position it is in at the time the CLOSE is given.

c) If the LOCK option is specified, all reels belonging to the file will be rewound except for those reels controlled by a prior CLOSE REEL, and the entire file cannot be read from or written upon.

CLOSE file-name 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 then also be rewound with interlock. The necessary processing will be performed.

When a CLOSE REEL is given, the locking and rewinding options of the CLOSE REEL, if used, will 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 will be executed whereever possible, for all mounted reels of the file, i.e., 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 the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION), the standard end of file processing is not performed whenever this file is not present, obviously.

When a file contains multiple reels, the opening and closing of individual reels is automatic, It is only required that the programmer close the file when processing is to be terminated. A CLOSE file-name-1 should be executed for each file that has been OPENed.

The WITH RELEASE option may only be used with CLOSE file-name, it has no meaning with a CLOSE file-name REEL option. The file released may not be opened again in the program; it is no longer available. The file is available to the supervisory control program so that the associated UNISERVO or UNISERVOS may be assigned to another program. If an output file has been designated as a SORT-FILE in the FILE-CONTROL paragraph of the Environment Division, then a CLOSE will cause the file to be sorted so as to appear in sequential order when subsequently OPENed for INPUT and read. (See Sorting, Section X.)

Example:

```
951100     CLOSE EDITED-SHIPMENTS WITH LOCK COST-OF-SALES-RATES WITH     7543004
951200     LOCK COSTED-SHIPMENTS WITH LOCK NOT-COSTED-ITEMS WITH LOCK    7543004
951500     LOSS-ITEMS WITH LOCK ERROR-LISTING WITH LOCK.                 7543004
```

(5) COPY

The function of the COPY statement in Procedure Division is to copy a procedure paragraph from the COBOL library.

The format is:

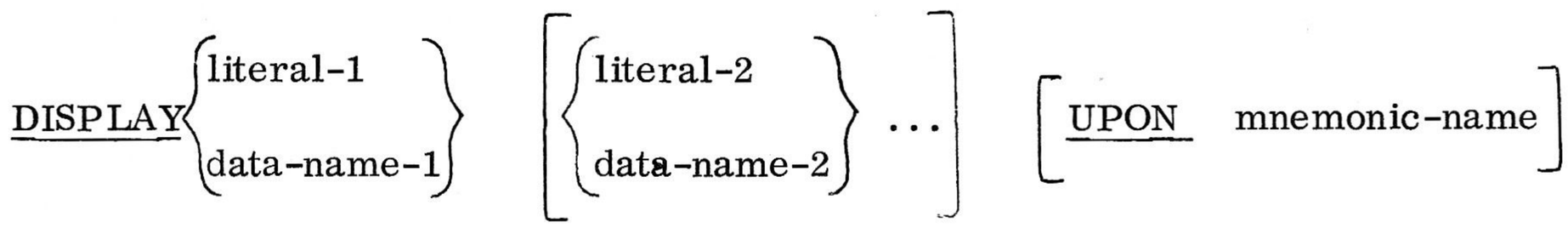
COPY procedure-name FROM LIBRARY

The procedure-name must be the name of a paragraph in the COBOL library. The paragraph containing the COPY statement must contain only the COPY statement. The entire contents of the paragraph in the COBOL library, except for its name, replace the COPY statement paragraph. The name of the procedure copied is the name given to the COPY paragraph. The result is the same as if the paragraph copied from the COBOL library had been written in place of the COPY statement.

This is a statement-for-statement exact copy.

(6) DISPLAY

(a) The function of the DISPLAY verb is to allow low volume data to be recorded upon an available hardware device. The format is:



The mnemonic-name corresponds to the Console type-writer or the paper-tape punch, as defined under the SPECIAL NAMES paragraph in the ENVIRONMENT DIVISION. For the UNIVAC 1107 the standard DISPLAY device is the CONSOLE and if the UPON option is not used the compiler will assume the CONSOLE (typewriter).

When the DISPLAY consists of multiple operands (data-names), the data comprising the first operand is stored as the first set of characters, the data comprising the second operand as the second set of characters, and so on, until the minimum unit is filled. This operation continues until all information is displayed. Data comprising an operand may extend into subsequent units.

Where an operator is expected to introduce variable data, the suggested programming technique is the ACCEPT should be preceded by a DISPLAY with instructions on what the ACCEPT should consist of and the format.

Example:

```
600800    DISPLAY "STORE PRESENT MONTH AS 2 DIGITS"          7543004
```

This will cause the literal which is bounded by quotation marks above, to be displayed upon the CONSOLE.

A combination of data-names and literals can be employed in a DISPLAY statement as indicated in the format. As an illustration, assume that the data-name TOTAL-AMT has been accumulated and it is desired to visually display this area, which contains the figure 049283.

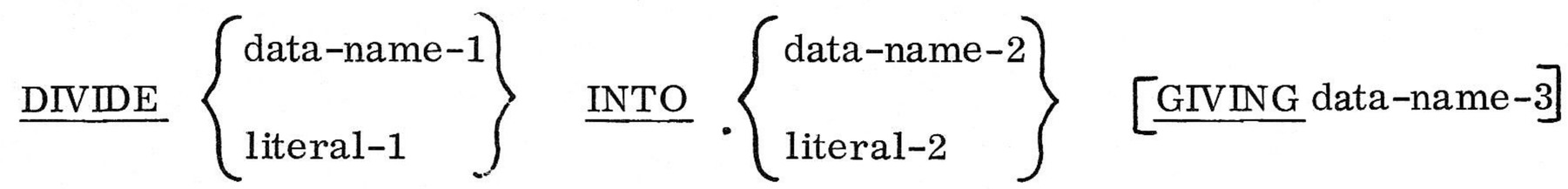
```
DISPLAY "GRAND TOTAL  "    TOTAL-AMT.
```

The information that will appear upon the CONSOLE is:

```
GRAND TOTAL    049283
```

(7) DIVIDE

The function of this verb is to divide one numeric value into another, setting the value of an item equal to the result. The format is:



In the format, data-name-1 is the divisor, data-name-2, the dividend, and data-name-3 the quotient. Where the GIVING option is not stated, the quotient replaces the value of the dividend (data-name-2). In this case a literal must not be used as the dividend.

The data-names used must reference numeric elementary items whose descriptions appear in the DATA DIVISION of the program.

All rules specified under the ADD verb regarding the ON SIZE ERROR option, the ROUNDED option, the GIVING option, truncation, and the editing of results, apply to the DIVIDE verb.

An error will be indicated at compilation time if the data description for either data-name-1 or data-name-2 specifies the presence of editing symbols. Literals used must be numeric.

Division by zero will result in a "size error"; the rules specified under the ADD verb with respect to ON SIZE ERROR apply.

The maximum size of the divisor is eighteen (18) digits. The maximum size of the quotient is eighteen (18) digits.

As an illustration, assume that three fields have been defined as computational with the following formats:

```
TONS PICTURE IS                 H99V9
HRS PICTURE IS                  H9V99
TONS-PER-HR PICTURE IS          H9V9
```

The following instructions can now be given:

```
DIVIDE HRS INTO TONS GIVING TONS-PER-HR
ROUNDED ON SIZE ERROR GO TO ERROR-LIST.
```

If TONS has a value of 68V4 and HRS has a value of 1V23, then control will pass to ERROR-LIST since the quotient (55V6) has a greater number of integral places than TONS-PER-HR allows. If TONS-PER-HR has a PICTURE of H99V9, then 55V6 will be stored at TONS-PER-HR and control will pass to the next procedure statement.

Immediately preceding the above instruction, the programmer probably will want to issue the statement IF HRS IS ZERO GO TO ... (Some predetermined procedure-name). This will check HRS for ZEROS thus obviating the possibility of an ON SIZE ERROR due to the divisor containing zeros.

(8) ENTER

There are two functions performed by the ENTER verb. The first is to provide a means of including UNIVAC 1107 relative object code (produced by some source code processor other than the COBOL compiler) into COBOL object programs. The second is to allow incorporation and use of independently compiled COBOL subprograms by the COBOL user.

Option 1:

ENTER routine-name SUBROUTINE [REFERENCING data-name-1[, data-name-2...]]

This option provides means for incorporating relative closed subroutines in a COBOL program before run time. Routine-name must be five (5) or fewer characters in length and the first character must be a non-blank alphabetic. The referencing option is used to name the argument or arguments of a given usage of the named subroutine. Upon encountering this sentence in a program, the COBOL compiler will generate a proper calling sequence referencing the named subroutine and, if the referencing option is used, descriptions of the data-items (arguments of the closed subroutine) will be included in the calling sequence. The named subroutine will be incorporated into the COBOL object program from a Subroutine Library prior to run time. During the running of the object program, this sentence will cause control to be transferred to the previously incorporated subroutine (which must be written according to the standard subroutine format) and will provide for return of control upon the completion of the execution of the subroutine.

Within any body of COBOL programming to be compiled at one time, any number of references may be made to a given closed subroutine using this option of ENTER. These will cause only one copy of the given subroutine to be included in the COBOL object program. All different subroutines used within a given COBOL program must have unique names.

Option 2:

ENTER { LOAD REFERENCING { subprogram-name / data-name } / entry-name PROCEDURE }

The concept of independently compiled subprograms is discussed at length in Section X.

This option enables the programmer to incorporate independently compiled COBOL subprograms into his COBOL program at run time, and to make cross references between such subprograms.

Considering the two functions of this option separately:

a. ENTER LOAD REFERENCING { subprogram-name / data-name }

1) Subprogram-name is the name given to the independently compiled subprogram, and within a given COBOL program subprogram-names must be unique.

2) If used, subprogram-name must be five (5) or fewer characters in length.

The case may arise that there are a large number of subprograms used by a COBOL program. The data-name option allows the programmer the facility of assigning an ordinary COBOL data-name to the set of names of such programs. This means that there must be an entry in the Data Division of the program which references this data-name, describing the contents of this data-name to be five (5) alphanumeric characters.

It is then possible to change the value of data-name, according to the requirements of the program, to be the name of any desired subprogram. Changing the value of data-name may be accomplished by any means provided in the COBOL language.

The function of ENTER LOAD REFERENCING is to cause the loading of the designated subprogram at run time. The subprogram will be located in UNIVAC 1107 memory in a subprogram area and there may be no more than one subprogram in that area at any given time. A program may contain as many ENTER LOAD REFERENCING statements as desired. If it is desired to incorporate independently compiled subprograms, there must be at least one ENTER LOAD REFERENCING statement in the incorporating program.

b. ENTER entry-name PROCEDURE

This option provides the means for transferring control between an independently compiled COBOL subprogram and the incorporating COBOL program. Entry-name must be five (5) or fewer characters in length. Entry-name is a paragraph-name in the referenced program and is the point to which control will be transferred. Entry-names within a subprogram must be unique with respect to entry-names in that subprogram and entry-names in the incorporating program. Observe that it is possible to have references in the subprogram to procedures in the incorporating (main) program.

Every distinct entry-name in a sub-program which is referenced by an ENTER entry-name PROCEDURE statement in the main program, must have a corresponding RETURN entry-name statement in a subprogram. Likewise, every distinct entry-name in the main program which is referenced by an ENTER entry-name PROCEDURE statement in a subprogram must have a corresponding RETURN entry-name statement in the main program. If the procedure ENTERed is in a section whose priority is greater than 49, then it is not permissable to transfer control to a section with a different priority (except priority zero) before executing the RETURN statement.

(9) EXAMINE

The function of the EXAMINE verb is to replace a given character and/or to count the number of times it appears in a data item. The data-name being EXAMINEd cannot have USAGE COMPUTATIONAL-1, i.e., binary. The format is:

```
EXAMINE data-name { TALLYING { ALL          } literal-1 [REPLACING BY literal-2] }
                  {          { LEADING      }                                   }
                  {          { UNTIL FIRST  }                                   }
                  { REPLACING { ALL          } literal-3 BY literal-4           }
                  {           { LEADING      }                                  }
                  {           { UNTIL FIRST  }                                  }
                  {           { FIRST        }                                  }
```

Any literal used in an EXAMINE statement must consist of only one character. If the description of data-name 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, any operational sign associated with the item will be ignored.

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

a. Occurrences of literal-1 when the ALL option is used.

b. Occurrences of literal-1 prior to encountering a character other than literal-1 when the LEADING option is used.

c. Characters not equal to literal-1 encountered before the first occurrence of literal-1 when the UNTIL FIRST option is used.

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

a. When the ALL option is used, then literal-2 (or literal-4) is substituted for each occurrence of literal-1 (or literal-3).

b. When the LEADING option is used, the substitution of literal-2 (or literal-4) terminates as soon as a character other than literal-1 (or literal-3) or the right-hand boundary of the data item is encountered.

c. When the UNTIL FIRST option is used, the substitution of literal-2 (or literal-4) terminates as soon as the first literal-1 (or literal-3) or the right-hand boundary of the data item is encountered.

d. When the FIRST option is used, the first occurrence of literal-1 (or literal-3) is replaced by literal-2 (or literal-4).

Note that the special register, TALLY, is limited to a length of five digits, and it may be used elsewhere in the program (i.e., ADD 3, TALLY). Literal-1, literal-2, literal-3, and literal-4 must be single character literals.

Example:

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

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

(10) EXIT

The function of the EXIT verb is to provide an end point for a procedure that is to be executed by means of a PERFORM verb. The format is simply:

EXIT

The 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 sub-routine; in this case, each of these points should come together at the EXIT in order to provide one ending point path.

EXIT will always 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 the related PERFORM, control will pass through the EXIT point to the first sentence of the paragraph following the PERFORM paragraph.

```
670130 SET-MON.                                                          7543004
670140     PERFORM MO-SEARCH THRU MO-END VARYING MO-CTR FROM 1 BY 1       7543004
670150     UNTIL LIMIT.                                                  7543004
.
.
.
671800 MO-SEARCH.                                                        7543004
671900     IF PRESENT-DATE IS EQUAL TO MO-CTR GO TO PUT-DATE.            7543004
672000 MO-END.                                                          7543004
672100     EXIT.                                                         7543004
672200 PUT-DATE.                                                        7543004
672300     MOVE MON (MO-CTR) TO MONTH-STORAGE IN SUB-HEADING-1.          7543004
672400     GO TO MO-END.                                                 7543004
```

In this example there are the following two possible exits from the MO-SEARCH routine: (1) if MO-CTR reaches the value of LIMIT, then control must pass to the sentence following line 670150; (2) If PRESENT-DATE becomes equal to MO-CTR, control will pass to PUT-DATE which will place the name of the present month in SUB-HEADING-1. It is still desired to return to the statement following 670150, however, and the paragraph MO-END is the method allowing this return from either of the two paths taken.

(11) GO TO

The function of the GO TO verb is to provide a means of departure from the normal sequence of procedures, i.e., it is used to specify a transfer. The format is:

Option 1:

GO TO [procedure-name-1]

Option 2:

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

DEPENDING ON data-name

When using option 1, if the GO TO statement is to be ALTERed, the paragraph in which the GO TO statement is included 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. If procedure-name-1 is omitted, and if the GO TO statement is not referenced by an ALTER statement prior to the first execution of the GO TO statement, an error stop provided by the compiler will occur at object time.

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

Example:

```
640200  CALCULATE.                                              7543004
640300      GO TO CK-FOR-TIN-WKS.                               7543004
```

The example previously 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.

```
651700  CHANGE-CALC.                                            7543004
651800      ALTER CALCULATE TO PROCEED TO CALCULATE-TONS.       7543004
651900      GO TO CALCULATE-TONS.                               7543004
```

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

The example above could have been effected 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. Upon determination that 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.

A GO TO statement can reference a NOTE paragraph. Control will be transferred to the NOTE paragraph, will be passed through to the paragraph following the NOTE paragraph.

(12) MONITOR

The function of the verb MONITOR is to provide dynamic printout of the values of specified items. The format is:

MONITOR data-name-1 [,data-name-2 . . .]

MONITOR writes the values of data-name-1, data-name-2, etc. referencing their names. In addition, the name of the current procedure paragraph is written.

The above output is written onto the file defined in the File Control paragraph of the Environment Division as having the mnemonic-name DIAGNOSTIC-FILE.

(13) MOVE

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

$$\underline{\text{MOVE}} \quad \left\{\begin{matrix}\text{literal}\\ \text{data-name-1}\end{matrix}\right\} \ \underline{\text{TO}} \ \text{data-name-2} \quad \left[\text{data-name-3 . . .}\right]$$

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

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

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.

When moving group items, the move is from left to right. If the item SIZE or USAGE is not identical a diagnostic message will be given. Truncation of low-order positions of the source item will occur if the receiving area is smaller. Space fill of low-order positions of the receiving area will occur if the source item is smaller. If the source group is computational and the receiving group is not, (or vice versa) a diagnostic message will be given and the above rule on size will govern the results of the move.

When both the receiving and source areas are elementary items, editing as specified in the receiving area will be automatically performed with the execution of each MOVE command. The rules governing this are as follows:

For NUMERIC elementary items

a. If the source area is larger than the receiving area, truncation of high-order positions will occur for non-decimal fields. (Truncation is the dropping off of excessive digits). If the receiving area is larger than the source area, zero-fill will be accomplished to the left, or high-order, positions for whole integers. 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 MOVE
9V9	12	99V99	0120
9V999	8765	V99	76

b. Data will be converted from the form shown in the source area to the form of the receiving area (e.g., numeric to alphanumeric, etc.).

c. Zeros will be replaced with blanks and the insertion of a dollar sign, a decimal point, commas, etc., with proper alignment, will be accomplished in accordance with the PICTURE of the receiving area. If these latter characters are in a source area, the field (s) will be non-numeric and, thus, MOVEment must conform to the non-numeric rules.

d. If no decimal point has been specified (either assumed or actual), data will be stored in the receiving area from right to left (right justification) unless JUSTIFIED LEFT or SYNCHRONIZED LEFT has been specified.

e. If a numeric item with an assumed decimal point is moved to a justified item a precautionary diagnostic will be produced and the source item will be moved as an integer.

For NON-NUMERIC 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 will be replaced with blanks (spaces).

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

Examples:

```
642000   MOVE A-FIELD TO COSTED-A-FIELD.          7543004
```

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

```
610500   MOVE 128 to CTR.                         7543004
```

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

```
621500   MOVE SPACE TO SIZE-CODE-SGN IN ERRORS. 7543004
```

The field named SIZE-CODE-SGN will be entirely filled with spaces

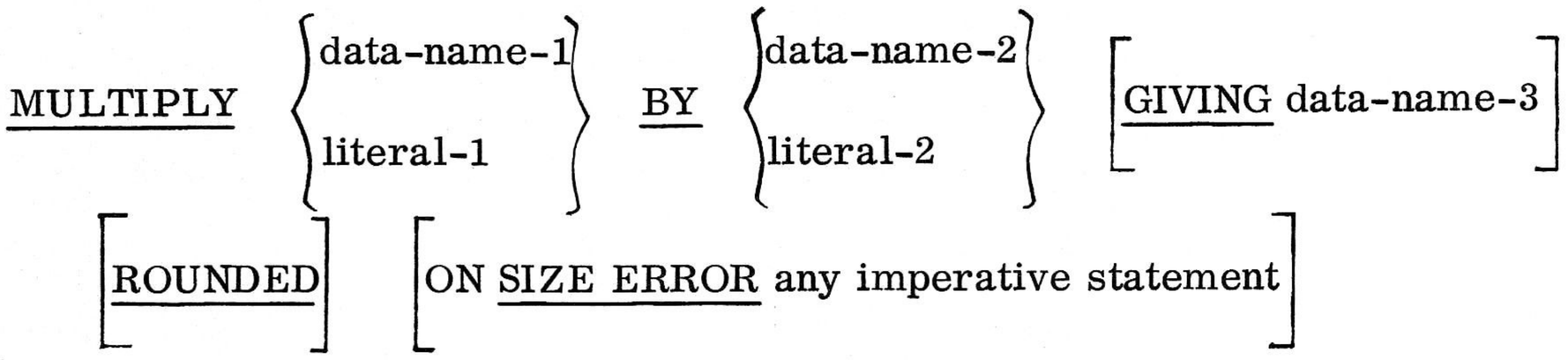
Examples of the behaviour of the MOVE verb.

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	$ZZZ9.99	$ 67.89

(14) MULTIPLY

The function of the MULTIPLY verb is to multiply two numerical data items and to substitute the resulting product for the value of an item.

The format is:



The data-names used must reference numeric elementary items whose descriptions appear in the DATA DIVISION.

All rules specified under the ADD verb regarding the ON SIZE ERROR option, the ROUNDED option, the GIVING option, truncation, and the editing of results, apply to the MULTIPLY verb.

The maximum size of the multiplier or multiplicand is eighteen (18) digits.

The formats of data-name-1 and data-name-2 may never contain editing symbols (e.g., dollar signs, commas, etc.). An error will be indicated at compilation time if the data descriptions for either data-name-1 or data-name-2 specifies the presence of editing symbols. Operational signs and implied decimal points are not considered editing symbols.

When the GIVING option is not used, the result will be placed in the multiplier (second operand) and, in this instance, the second operand must not be a literal.

When the GIVING option is used, data-name-3 will contain the product of the multiplication. Data-name-3 may contain editing symbols.

Example:

```
690900    MULTIPLY TCI-RATE IN RATES (CTR) BY BASE-BOXES IN B-FIELD    7543004
691000    GIVING TCI-COST IN C-FIELD ROUNDED.                          7543004
```

The product of BASE-BOXES IN B-FIELD and TCI-RATE IN RATES will be stored at TCI-COST IN C-FIELD with rounding taking place. BASE-BOXES has been defined as a whole number (no decimals); TCI-RATE has four decimal places, and TCI-COST has only two decimal places. The product will contain four decimal places, but will be ROUNDED to two for storing in the TCI-COST field.

The purpose of the subscript (CTR) is to identify which TCI-RATE is to be used in the multiplication, since these rates are in a table. Prior to the execution of the above instruction, the value of CTR has been established through the use of a table look-up.

(15) NOTE

The function of the NOTE verb is to allow the programmer to insert comments in the source program for explanatory purposes. These comments will be listed but have no effect on the object program. The format is:

NOTE comment-1 [comment-2. . .]

If NOTE is the first word of a paragraph, the entire paragraph is commentary. That is, a paragraph-name would appear on the line preceding the NOTE sentence(s), and another paragraph-name would appear immediately following the NOTE sentence(s). NOTE may be, however, the first word in a sentence appearing in any procedural paragraph in which case the NOTE continues to the next period.

Any number of sentences can be written in a NOTE paragraph as program execution commences with the paragraph following the NOTE paragraph.

Any combination of characters from the COBOL character set except a period which is the last character on the card or which is followed by a blank can follow the verb NOTE.

Example:

```
601950 NOTE2.                                                          7543004
601955      NOTE IF ABOVE STATEMENT IS TRUE THE ITEM HAS BEEN PRECOSTED. 7543004
601980 COMPARE-FIRST-DIGIT-WKS.                                        7543004
```

The above is an example of a one sentence paragraph (NOTE 2 and COMPARE-FIRST-DIGIT-WKS being paragraph-names). NOTE 2 can be any paragraph-name. NOTE 2 will not appear in the object program but will appear in the program listing.

(16) OPEN

The function of the OPEN verb is to initiate the processing of both input and output files. The format is:

$$\underline{\text{OPEN}} \left\{ \begin{matrix} \underline{\text{INPUT}} \\ \underline{\text{OUTPUT}} \end{matrix} \right\} \left[\text{WITH} \left[\underline{\text{NO}} \right] \underline{\text{REWIND}} \right] \quad \text{file-name-1} \quad \left[\text{, file-name-2...} \right]$$

Any FD entry 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 in addition to 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 (which has been defined in the FD entry); similarly, if OUTPUT has been specified, the OPEN statement causes the writing of the label upon the output 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.

The OPEN does <u>not</u> 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, the object program will ask the operator, via the Console typewriter, whether the file is absent or present. If the reply to the interrogation is negative, i.e., the file is not present, the file will not be OPENed, an indication

of the absence of the file will occur, and an "end of file" signal will be sent to the input-output control system of the object program. Thus, when the first READ for this file is encountered, the AT END path specified in the READ statement will be taken.

If the WITH REWIND option is used, the file will be rewound.

If the WITH NO REWIND option is used, the file will not be rewound. This provides an OUTPUT file the ability to stack files and the ability on INPUT WITH NO REWIND to read those stacked files.

If neither of the REWIND options is stated, the file will not be rewound.

If OPEN INPUT SORT-FILE is stated a previous OPEN OUTPUT SORT-FILE and a CLOSE OUTPUT SORT-FILE must have been stated previously. The items received from the OPEN INPUT SORT-FILE will be received by the program by a READ in sequential order (i.e., sorted).

```
600500     OPEN INPUT EDITED-SHIPMENTS, COST-OF-SALES-RATES.          7543004
600600     OPEN OUTPUT COSTED-SHIPMENTS, LOSS-ITEMS, ERROR-LISTING.   7543004
```

Label checking will be performed on EDITED-SHIPMENTS and COST-OF-SALES-RATED files, and label writing executed for COSTED-SHIPMENTS, LOSS-ITEMS and ERROR-LISTING files according to label specifications for these files in the DATA DIVISION. Note that each of the operands is an FD name from the DATA DIVISION. It is necessary to use two separate OPENs as shown above in referencing input and output files.

(17) PERFORM

The function of the PERFORM verb is to allow for 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 to provide an automatic return to the normal sequence. There are four different formats, which vary in complexity as follows:

Option 1:

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

Option 2:

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

Option 3:

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

Option 4:

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

General

PERFORM is a means by which program loops can be written in COBOL. The loop may be executed once or a number of times, as determined by a variety of controls.

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:

a. 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.

b. 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.

c. 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.

d. 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.

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

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

Moreover, a procedure referenced by one or more PERFORMS can also be executed by "falling through", that is, by entering the procedure through the normal passage of control from one statement to the next in sequence. Normally, procedure-name-1 should not be the next statement after the PERFORM. If it were the next statement, the procedure probably would be executed one more time than was intended because, after execution of the PERFORM, control would pass to procedure-name-1 in the normal continuation of sequence.

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 had existed.

Bypasses

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 will be initially set to the next procedural statement (i.e., GO TO SUBROUTINE-2). Upon execution of MAIN-ROUTE, the following steps will occur:

a. The line number of the statement following the PERFORM (i.e., line 23) will be placed in BYPASS-1.

b. Control will be transferred to SUBROUTINE-1.

c. At the completion of SUBROUTINE-1 (lines 1 through 3), BYPASS-1 will be 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. Thus, the paragraph-name SUBROUTINE-3 was inserted to stop the PERFORM loop after execution of SUBROUTINE-2.

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. 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 brought to a successful conclusion 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.

Care must be taken in using a conditional (IF) statement in routines referenced by a PERFORM. For example, assume that on line 4 in the previous illustration, there was the instruction IF A IS LESS THAN B, GO TO ABLE. Thus, when GO-AGAIN was executed, BYPASS-2 would have the address of line 57 plugged into it. However, if A was less than B, control would pass to ABLE and BYPASS-2 would not be reset. Suppose that control subsequently returned to SUBROUTINE-1 (either on an unconditional transfer of control or on a "falling through" situation). This time if A was not less than B, control would pass to line 57 from BYPASS-2, which would probably not be the path intended by the programmer. However, if the conditional statement is contained within a closed sub-routine (a routine reached only through use of a PERFORM), then the bypass is always

under specified control. For example, assume that SUBROUTINE-1 THRU SUBROUTINE-2 was PERFORMED from several different places throughout the program and there was no GO TO statement referencing either of these procedure-names, nor was it possible to "fall through" to these instructions; upon performance of GO-AGAIN, BYPASS-2 would be set to return control to line 57. If A was less than B, control would pass to ABLE as above. When line 74 was executed, BYPASS-2 would be changed to transfer control to line 75. Regardless then of the outcome of the A and B comparison, BYPASS-2 contains the procedure-name for proper return placed there by each PERFORM in process.

Essentially then, the programmer has the facility to use conditional sentences 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

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:

```
661700     WR-EX.                                    7543004
661800        WRITE EXCEPTIONS.                      7543004
661900     MV-1-EX-PG.                               7543004
   .
   .
   .
670400        PERFORM WR-EX.                         7543004
```

The PERFORM WR-EX will cause the paragraph WR-EX to be executed once and control passed to line 670500. All necessary program steps to accomplish this will be generated by the compiler.

Option 2

Option 2 is the TIMES option. This form provides a means of performing a procedure repetitively a specified number of times. The number of times, whether stated as integer-1 or as data-name-1, 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.

Illustration:

```
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 would be executed four times and then control would pass to the statement following the PERFORM.

Option 3

Option 3 is the UNTIL option. This option is essentially the same as the TIMES option, except that the PERFORM causes an evaluation of the specified conditional expression instead of counting and testing the count against a specified integer.

Condition-1 may be any conditional statement; that is, the condition may involve relations and tests. Condition-1 is evaluated before execution of procedure-name-1. If the condition is false, control is passed to procedure-name-1, the procedure is executed once and control returns for another evaluation of condition-1. 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 "drops through" the PERFORM instruction to the next sequential sentence.

Illustration:

```
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 data-name option (Option 4) is used to PERFORM a procedure repetitively, increasing or decreasing the value of data-name-2 for each repetition, until a specific conditional expression (condition-1) is satisfied. Only one data-name can be varied per PERFORM statement using this option. Option 4 is assumed to be arithmetic in nature and the arithmetic rules apply.

The PERFORM mechanism first sets the value of data-name-2 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 data-name-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 is determined to be true at which point control is passed 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). These values may be positive, negative or zero. It should be noted that after the condition is found to be true, data-name-2 will be one increment (or decrement) greater than its last used value unless the starting value (FROM value) is zero.

Example:

Data Division

```
351200 77   MO-CTR PICTURE IS 99 USAGE IS COMPUTATIONAL.                    7543004
351300       88  LIMIT VALUE IS 13.                                         7543004
351400 77   PRESENT-DATE PICTURE IS 99 USAGE IS COMPUTATIONAL.              7543004
   .
   .
   .
510500 01   MONTH-TABLE PICTURE IS A (36)                                   7543004
510600       VALUE IS "JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC".               7543004
510700 01   MONTH-LOOK REDEFINES MONTH-TABLE.                               7543004
510800       02  MON PICTURE IS AAA OCCURS 12 TIMES                         7543004
```

Procedure Division

```
670140     PERFORM MO-SEARCH THRU MO-END VARYING MO-CTR FROM 1 BY 1        7543004
670150     UNTIL LIMIT.                                                    7543004
   .
   .
   .
671800   MO-SEARCH.                                                        7543004
671900     IF PRESENT-DATE IS EQUAL TO MO-CTR GO TO PUT-DATE               7543004
672000   MO-END.                                                           7543004
672100     EXIT.                                                           7543004
672200   PUT-DATE.                                                         7543004
672300     MOVE MON (MO-CTR) TO MONTH-STORAGE IN SUB-HEADING-1.            7543004
672400     GO TO MO-END.                                                   7543004
```

Note that there is more than one possible exit from the MO-SEARCH routine, as follows:

a. If MO-CTR reaches the value of LIMIT (13); or

b. If PRESENT-DATE equals MO-CTR.

If either of these conditions is satisfied, control is to be returned to the sentence following line 670150. Thus, MO-END was used as procedure-name-2, and consists only of the EXIT verb.

Assume that PRESENT-DATE contains the number 03. Upon execution of line 670140, a 1 (the FROM value) will be placed in MO-CTR. The conditional sentence IF LIMIT GO TO MO-END will be generated at compilation time and evaluated at object time. (This sentence, through the use of the condition-name LIMIT, is the same as IF MO-CTR IS EQUAL TO 13 GO TO MO-END.) Since MO-CTR is 1, the test will fail and MO-SEARCH will be performed. PRESENT-DATE (3) is not equal to MO-CTR (1) so MO-CTR will be increased by 1 (the BY value) and control looped back to the comparison of MO-CTR and LIMIT. This process will continue until MO-CTR has been incremented to the number 3, and then, upon execution of MO-SEARCH, control will pass to PUT-DATE. At the completion of PUT-DATE control is transferred to the EXIT (MO-END) and thence to the sentence following line 670150.

If PRESENT-DATE had contained any number greater than 12, MO-SEARCH would be performed only twelve times. At the completion of the twelfth loop, MO-CTR would be incremented again and contain the value 13. Then the test would be made for LIMIT, found to be true, and control would pass to MO-END (the EXIT) without PUT-DATE ever being executed.

The preceding is an example of Option 4. Procedure-name-1 and procedure-name-2 can be any allowable paragraph while condition-1 may be any allowable

conditional statement. Note again, that UNTIL means "until but not including".

As an example, when MO-CTR, in the preceding example, has a value between 1 and 12, MO-CTR will be incremented by 1. When MO-CTR has a value of 13 (the stated value of LIMIT) MO-CTR will not be incremented, and the PERFORM command will not be executed.

NESTING

"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.
 .
 .
 .
74.  PERFORM 29 THRU 31.
 .
 .
 .
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.
```

18) READ

The function of the READ verb is to make available the next logical record from an input file and to allow performance of a specified imperative statement when end of file is detected. The format is:

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

File-name must be an FD described in the DATA DIVISION.

An OPEN statement for a file must be executed before the first READ command is given, and both the OPEN and a READ must be given before referencing any data in the file. The OPEN will check the label and position 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 an item-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 same memory area. Thus, if there is more than one 01 entry in a given FD, a determination must be provided by the programmer so that reference is made only to information present in the current record. As an illustration, assume that HEADER and DETAIL are record description 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 advances the next record of MASTER, then IDENTITY must be interrogated to determine whether the record is a HEADER or DETAIL item. If this is not done, then a command directed to PART-CODE will at times, (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 data-name option is used, the input file is read and then moved to data-name, which must be the name of a working-storage, common-storage, or output record area. In this instance, moving will occur according to the rules specified for the MOVE verb. Under this option, "file-name RECORD" is available in the input record area as well as in the INTO area.

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

a. If labels are present (as defined in the FD) the standard end-of-reel label subroutine is performed.

b. A tape swap is effected.

c. If labels are present, the standard beginning reel label subroutine is executed.

d. The next logical record of the file is made available.

The AT END option must be written at least once for all tape files. If just one of the READ sentences referencing a given file contain an AT END statement, the compiler will, in effect, append that statement to each of the other READ sentences referencing that file. If more than one AT END is used for a given file, then all READS

directed to that file must have an AT END statement. If this is not done, an error will be indicated at compilation time. After execution of the "any simple imperative statement" an attempt to perform a READ without the execution of a CLOSE and a subsequent OPEN for that file will constitute an error in the object program.

The "AT END" option may never be used with a READ statement which references a peripheral input file (i.e., files assigned to a card reader). Recognition of end-of-file conditions and procedures for handling end-of-file conditions for such files must be accomplished by statements in the PROCEDURE DIVISION of the COBOL program. (i.e. programming)

If the file-name has been specified as OPTIONAL in the ENVIRONMENT SECTION and is not present at the object time, the "any simple imperative statement" will be executed when the first READ for the file is encountered.

Example:

```
600900  READ COST-OF-SALES-RATES RECORD AT END GO TO RATE-EOF.  7543004

692100  READ COST-OF-SALES-RATES RECORD                         7543004
```

In the above, the intended AT END path for all READ sentences for COST-OF-SALES-RATES is the same. Therefore, it is not necessary to give an AT END for each READ; one will suffice. Suppose line 692100 was the only READ for COST-OF-SALES-RATES; then, an error would be indicated by the compiler since there is no explicit AT END statement applying to this file-name. If COST-OF-SALES-RATES has been described as having one record per block, then each execution of the above sentences will cause a physical movement (READ) of the tape. If more than one record per block, then the item-advance feature of the READ will be employed.

19) RETURN

The function of the RETURN verb is to inform the compiler that the paragraph-name referenced by the RETURN statement will be ENTERed from an Independently Compiled Program. (See Special Features, Section X, and ENTER, previously defined in this section). The format is:

RETURN entry-name

The entry-name must be the paragraph-name of the statement ENTERed.

For example:

Main program

ABC. ENTER XYZ PROCEDURE.

———
———
———

Independently Compiled Subprogram

XYZ. ———
———
———

RETURN XYZ.

In the preceding example, the main program used the ENTER XYZ PROCEDURE statement to transfer control to XYZ PROCEDURE, the Independently Compiled Subprogram. The RETURN XYZ statement causes the entrance to XYZ to be altered to store the proper return to the main program (the statement following the ENTER XYZ PROCEDURE statement) in RETURN XYZ. Since it is possible to use the ENTER entry-name PROCEDURE statement in a subprogram to refer to the main program, this entire example would also be correct with "main program" and "independently compiled subprogram" interchanged.

The entry-name used in common by the ENTER XYZ PROCEDURE and RETURN XYZ statements (i.e., XYZ) must not exceed 5 characters in length and the first character must be a non-space alphabetic.

20) STOP

The function of the STOP verb is to halt the object program either permanently or temporarily. The format is:

$$\underline{\text{STOP}} \left\{ \begin{matrix} \text{literal} \\ \underline{\text{RUN}} \end{matrix} \right\}$$

Either a literal or the key word RUN must be used with STOP. If a literal is employed, it will be displayed by the object program at the time STOP occurs upon the console printer. If the operator should elect, continuation of the object program will begin 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 statement of the program.

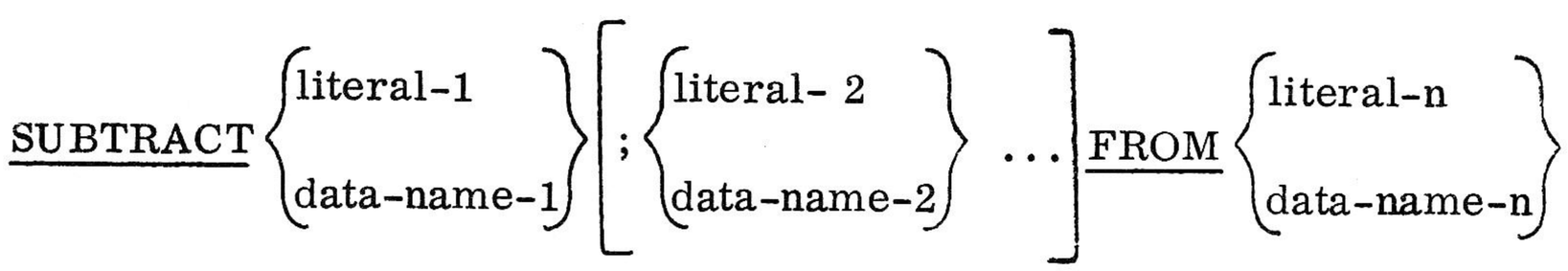
Some examples of the use of STOP are:

```
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.

21) SUBTRACT

The function of the SUBTRACT verb is to subtract one or more numerical data items from a specified item substituting the resulting difference for the current value of an item. The format is:



[GIVING data-name-m] [ROUNDED]

[ON SIZE ERROR any imperative statement]

All data-names used must reference numeric elementary items whose descriptions appear in the DATA DIVISION of the source program.

All rules specified under the ADD verb with respect to the ON SIZE ERROR option, the size of operands, the ROUNDED option, the GIVING option, truncation, and the editing of results, apply to the SUBTRACT verb.

Example:

```
650500  SUBTRACT USS-INV-COST IN COSTED-B-FIELD FROM TCI-COST IN  7543004
650600  C-FIELD GIVING DENORMALIZING-VARIANCE.                    7543004
```

When this statement is executed, USS-INV-COST IN COSTED-B-FIELD is subtracted from TCI-COST IN C-FIELD and the resulting difference is stored at DENORMALIZING-VARIANCE. The sign of DENORMALIZING-VARIANCE is set according to the rules of algebra for subtraction. Both the subtrahend and the minuend are unaffected in the above example.

22) WRITE

The function of the WRITE verb is to release a logical record for an output file and to allow for vertical positioning if the output medium is an on-line printer. The format is:

```
WRITE record-name [ FROM data-name-1]

[ { AFTER  }                { data-name-2 }        ]
[ { BEFORE } ADVANCING      { integer     } LINES  ]
```

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. Memory space is allocated for the output when the file is OPENed, and failure to OPEN the file will result in severe damage to the memory contents when output is moved to a non-existent output area.

When the WRITE is executed, record-name is released for the output file, and thus, is no longer available. Therefore, all the instructions referencing this record must be performed before the WRITE occurs.

The FROM data-name-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. Data-name-1 must be the name of a constant area, a working-storage area, a common-storage area, or an input record area. If the format of data-name-1 differs from that

of record-name, the data is moved in accordance with the rules for the MOVE verb. The information in record-name is no longer available, the data of data-name-1 continues to be accessible. The names of data-name-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 this option:

a. When data-name-2 is used, it must have a positive integral value. The compiler will provide the mechanism in the object program to position the printer page according to the current value of data-name-2.

b. When integer is used, it must be a positive integral literal. The compiler will provide the mechanism in the object program to advance the printer page integer lines.

c. If integer is 0, there will be no line advance, and the next WRITE statement will cause printing on the same line.

   The ADVANCING option takes precedence over the LINE-SPACING option of the VALUE clause in the FD entry.

d. If integer has a value which would cause the line to be printed below the limit implied by the LINES-AT-BOTTOM option of the VALUE clause in the FD entry, printing will commence on the first line of the next page, after spacing the number of lines indicated by the LINES-AT-TOP option.

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

a. The standard end-of-reel label subroutine if labels are specified in the FD of the file.

b. A tape swap.

c. The standard beginning-of-reel subroutine, if labels are specified.

Examples:

```
651500 WRITE COSTED-DETAIL-SHIPMENTS.                          7543004
691300 WRITE SHIPMENTS-NOT-COSTED FROM DETAIL-SHIPMENTS.       7543004
```

ENVIRONMENT DIVISION

1. GENERAL DESCRIPTION

a. Overall Approach

The basic approach to the ENVIRONMENT DIVISION is to centralize those aspects of the total data processing problem which are dependent upon the physical characteristics of a specific computer. It provides a linkage between the logical concepts of data and records, and the physical aspects of the files on which they are stored.

The ENVIRONMENT DIVISION must be rewritten each time a given problem is run on a different computer. It has been included in the COBOL System to provide a standard way of expressing the computer dependent information which must be included as a part of the overall definition of every data processing problem to be run on a computer.

b. Organization

The ENVIRONMENT DIVISION has been divided into two sections - CONFIGURATION and INPUT-OUTPUT.

The CONFIGURATION Section, which deals with the overall specifications of computers, is divided into three paragraphs. They are: the SOURCE-COMPUTER, which defines the computer on which the COBOL Compiler is to be run; the OBJECT-COMPUTER, which defines the computer on which the program produced by the COBOL Compiler is to be run; and SPECIAL-NAMES, which relate the actual names of the hardware used by the COBOL Compiler to the names used in the program.

The INPUT-OUTPUT Section deals with the definition of the external media and information needed to create the most efficient transmission and handling of data between the media and the object program. This section is divided into two paragraphs. They are the I-O-CONTROL, which defines special input/output techniques, rerun, and multiple file tapes; and FILE-CONTROL, which names and associates the files with the external media.

c. Structure

The following is a general outline of the Sections and Paragraphs under the ENVIRONMENT DIVISION.

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  computer-name.
OBJECT-COMPUTER.  computer-name....
SPECIAL-NAMES.  device-name.
INPUT-OUTPUT SECTION.
FILE-CONTROL.  SELECT....
I-O-CONTROL.  APPLY....
```

Each of the five paragraphs of the Environment Division are discussed in the sections that follow.

1) SOURCE-COMPUTER

The source computer is assumed to be known at compilation time. This assumption is based on these facts:

a) The COBOL compiler has access to, or can itself determine the equipment configuration at compilation time.

b) The Executive Routine contains the parameters describing the computer it is controlling.

This paragraph serves only to clarify the computer for which the COBOL program was prepared.

The format is:

```
SOURCE-COMPUTER.  UNIVAC-1107.
```

2) OBJECT-COMPUTER

The function of the OBJECT-COMPUTER paragraph is to describe the computer upon which the program is to be run.

The Format is as follows:

OBJECT-COMPUTER. UNIVAC-1107

[WITH SUPERVISOR CONTROL]

[MEMORY SIZE integer WORDS]

[[literal-1] hardware-name-1 [, [literal-2]

hardware-name-2 ...]] .

The literals used must be unsigned (positive) numeric literals and may be omitted if equal to 1.

The computer name provides an automatic definition of a particular equipment configuration assumed by the COBOL compiler. If no qualifying phrases occur, the configuration is assumed to be that of the Source Computer.

This assumed configuration may be changed in the following way:

> By specifying the object computer in the accompanying qualifying phrases in this OBJECT-COMPUTER paragraph.

The assumed object computer configuration may specify more equipment than is needed. This OBJECT-COMPUTER paragraph would be used to describe the parameters of the run to the compiler. If the values specified in this paragraph are less than the minimal requirements of the run as specified in the File Control paragraph, then an error indication will be given during compilation.

All UNIVAC 1107 programs run under the control of the Executive Routine. Therefore, the WITH SUPERVISOR CONTROL is a redundancy and is here stated only for compatibility.

The MEMORY SIZE phrase is used to inform the compiler of the maximum memory permissible for the object program being produced. Integer-1 may be less than the object computer memory size, but it cannot be greater. Should the memory requirements of the program being compiled exceed the integer supplied with the MEMORY SIZE phrase, a diagnostic warning will be given. The compilation will proceed, however, and the programmer will be able to determine what course of action is required.

The MEMORY SIZE phrase does not include memory requirements for the Executive Routine or I/O Pools. As stated in the WITH SUPERVISOR CONTROL phrase all UNIVAC 1107 programs are assumed to run under Executive control.

The MEMORY SIZE clause specified in programs using the Sorting routine will include the area requirements (both data and program) of the Sorting routines.

The following fixed hardware names may be used to specify an Object-Computer configuration:

PRINTER(S)
CARD-READER-EIGHTY
CARD-READER-NINETY
CARD-PUNCH-EIGHTY
CARD-PUNCH-NINETY
CARD-READER-EIGHTIES
CARD-READER-NINETIES
CARD-PUNCH-EIGHTIES
CARD-PUNCH-NINETIES
DRUM
DRUMS
DISC-FILE
DISC-FILES
UNISERVO-II
UNISERVO-IIS
UNISERVO-III
UNISERVO-IIIS
UNISERVO-IIIC
UNISERVO-IIICS
TAPE-PUNCH
TAPE-READER

3) SPECIAL-NAMES

The purpose of the SPECIAL-NAMES paragraph is to provide a means of relating hardware with mnemonic names. The paragraph is not required if mnemonic names are not used in the Procedure Division or the Input/Output Section of the Environment Division.

The format is as follows:

[SPECIAL-NAMES. Hardware-name-1 IS

mnemonic-name-1

[, hardware-name-2 IS mnemonic-name-2 . . . .]]

Some hardware-names may have assigned to them unique mnemonic names which may be used with the ACCEPT and DISPLAY verbs.

A mnemonic-name cannot be written in any source program except where formats specifically show that the use of mnemonic-name is permitted.

Fixed hardware assignments must be accomplished in this paragraph. Assignment of a file to a fixed hardware device in the FILE CONTROL paragraph requires the use of a mnemonic-name in the FILE CONTROL paragraph. The mnemonic-name used is equated to a specific hardware device in this SPECIAL-NAMES paragraph.

The specific hardware names for the UNIVAC 1107 COBOL are defined and specified as follows:

a) Mnemonic-names equated to the following hardware names may be specified only in the Input-Output Section of the Environment Division:

CARD-READER-EIGHTY	k
CARD-READER-NINETY	k
CARD-PUNCH-EIGHTY	k

CARD-PUNCH-NINETY	k
PRINTER	k
UNISERVO-III	k
UNISERVO-II	k
UNISERVO-IIIC	k

k is an alphabetic character, A through Z, depending on the number of units in the user's configuration.

b) The standard input/output device for UNIVAC 1107 used with ACCEPT DISPLAY for small volume data is the hardware-name CONSOLE. A mnemonic-name equated with CONSOLE can be used with ACCEPT and DISPLAY statements (q.v.).

c) TAPE-READER: meaning the paper tape Reader. This device may be referenced only in an ACCEPT statement for which the FROM option specifies a mnemonic-name which is equated in this paragraph with the hardware-name TAPE-READER.

d) TAPE-PUNCH: meaning the paper tape punch. This device may be referenced only in the DISPLAY statement that contains an UPON option and the mnemonic-name used is equated in this paragraph with the hardware-name TAPE-PUNCH.

4. FILE-CONTROL

The function of the FILE CONTROL paragraph is to name each file, identify its medium, to allow particular hardware assignments, to specify alternat input/output areas and to facilitate multi-programming.

The format for the FILE-CONTROL paragraph is as follows:

FILE-CONTROL. SELECT [{ OPTIONAL / SORT-FILE }] { DIAGNOSTIC-FILE / file-name-1 } [RENAMING file-name-2]

ASSIGN TO [integer-1] { mnemonic-name-1 / hardware-name-1 } [, { mnemonic-name-2 / hardware-name-2 } ...]

[FOR MULTIPLE REEL] [, RESERVE { integer-2 / NO } ALTERNATE { AREAS / AREA }].

[SELECT ...]

This paragraph consists of a series of sentences each of which begins with the key word SELECT and ends with a period.

The name of each file which is used in a program must appear as file-name-1 in the FILE-CONTROL sentence following the verb SELECT once and only once. The name of each SELECTed file (i.e., file-name-1) must be unique within a program. If the RENAMING option is not used, then file-name-1

must have a File Description in the DATA DIVISION of the source program; if the RENAMING option is used, then file-name-2 must have a File Description in the DATA DIVISION of the source program.

It should be further noted that a file is not SELECTed simply because it appears after the word RENAMING in the FILE-CONTROL paragraph.

The key word OPTIONAL is required for input files which will not necessarily be present each time the object program is to be run.

The key word SORT-FILE is required for any file to be sorted by the Sort Routine. SORT-FILE must appear immediately following the word SELECT. Files specified for SORT-FILE must have the SEQUENCED ON clause in the File Description entry in the DATA DIVISION (Section VI).

The key word DIAGNOSTIC-FILE must be used in one SELECT sentence whenever the MONITOR verb is used (see Procedure Division, Section VII). DIAGNOSTIC-FILE must follow the word SELECT and the only other phrase required is ASSIGN. The DIAGNOSTIC-FILE must not have an FD entry since such an entry has no meaning.

If more than one file utilizes the same File Description, the RENAMING option must be included in the FILE-CONTROL paragraph. That is, the RENAMING option must be included if file-name-1 utilizes the File Description written for file-name-2; e.g., when a file is to be processed both as an input and as an output in the same program. RENAMING file-name-2 implies the sharing of a single File Description and does not allow these files to be referenced interchangeably in the program. If DIAGNOSTIC-FILE is used rather than file-name-1, the RENAMING clause must not be used.

All files used in the program must be ASSIGNed to an input or output medium (hardware-name). Hardware-name may be one of:

CARD-READER-EIGHTY
CARD-READER-NINETY
CARD-PUNCH-EIGHTY
CARD-PUNCH-NINETY
PRINTER
UNISERVO-II
UNISERVO-IIS
UNISERVO-IIIC
UNISERVO-IIICS
DRUM

For the UNISERVO-III magnetic tape, any of the following formats may be used:

UNISERVO-III
UNISERVO-IIIS

For specific peripheral units:

mnemonic-name

The mnemonic-name must be a mnemonic-name for a specific card reader, card punch, printer or magnetic tape unit when more than one of these units are on a given UNIVAC 1107 configuration and a fixed unit assignment is being made.

Files may not be ASSIGNed to Paper Tape peripherals, however, the ACCEPT and DISPLAY statements may reference these peripherals through the FROM and UPON options.

Integer-1 may only be specified when hardware-name-1 is UNISERVO-III and must be an unsigned (positive) numeric literal. It indicates the number of UNISERVOs to assign to file-name-1. If integer-1 is omitted, one UNISERVO III will be assigned.

The MULTIPLE REEL option must be included when:

a) Integer-1 is not specified and more than one reel may exist in a file.

b) Integer-1 is specified, but may be less than the total number of reels in a file.

The RESERVE statement allows the user to specify the number of storage areas to be allocated to the file. Integer-2 represents the number of additional areas desired, over and above the required minimum of one. It may never be zero. For those files where this phrase is used but where no additional storage areas are desired, the word NO is used instead of integer-2.

Example:

```
INPUT-OUTPUT SECTION.
FILE-CONTROL. SELECT EDITED-SHIPMENTS
  ASSIGN 2 UNISERVO-IIIS FOR MULTIPLE REEL
  RESERVE 1 ALTERNATE AREA.
```

5) I-O-CONTROL

The function of I-O-CONTROL is to specify the Input/Output technique and the points at which RERUN is to be established.

This section does not apply when SORT-FILE or DIAGNOSTIC-FILE options of the FILE-CONTROL Section have been specified. Each of these has self-contained I/O procedures.

The format for I-O-CONTROL paragraph is as follows:

```
[I-O-CONTROL.

 [[APPLY {DEMAND       }     ON [file-name-1] [, file-name-2 ...]]
         {STAND-BY     }
         {TRANSLATION  }
         {NO TRANSLATION}

  [; APPLY...]  [RERUN ON file-name-3 EVERY integer-1 RECORDS OF

                                                 [file-name-4]]]
```

The APPLY clause may be used more than once in an I-O-CONTROL paragraph. However, file-names used in one APPLY clause cannot also be used in another.

If the APPLY clause is omitted, the compiler will utilize the DEMAND system. If more than one file-name is specified within one APPLY clause, these files may be pooled in their input or output area assignments. Both an input file and output file must not be placed in the same pool.

DEMAND... Tnis system reads an input block into a memory area. For magnetic tape files any number of segment separators may appear on the input tape and are ignored when DEMAND is used.

When DEMAND is used and more than one output file for the UNISERVO IIIs is specified in a single APPLY clause, the output file areas will be pooled. The block areas within the pool will be equal in size and equal to the largest block size of any output file using the pool. Record sizes for these files need not be equal in size, as record handling is independent.

A separate area is required for input and output of the same file. Data records must be moved into output file areas before they can be written.

STAND-BY technique is the same as DEMAND technique except that two blocks are assigned to each file instead of one.

TRANSLATION or NO TRANSLATION is applicable to card files only.

If RERUN ON is specified, it is necessary to indicate when a rerun point is to be established and where the memory dump is to be written. Memory dumps may be written on any output file that has been assigned to a UNISERVO III except a SORT-FILE. The dump will be enclosed with the standard UNIVAC 1107 by-pass sentinels to distinguish the memory dump from the data on the file. File-name-3 specifies the output file which is to contain the memory dump.

File-name-4 must always be specified and together with integer-1 establishes the rerun points. File-name-4 may be either an input or output file. File-name-3 must always be an output file.

Care should be taken that file-name-3 has been OPENed for output before attempting to use it as the RERUN dump file.

Example of I-O-CONTROL entry:

```
I-O-CONTROL.

        APPLY DEMAND ON EDITED-SHIPMENTS
        COST-OF-SALES, APPLY DEMAND ON
        LOSS-ITEMS; RERUN ON ERROR-LISTING
        EVERY 10000 RECORDS OF EDITED-
        SHIPMENTS.
```

IX. IDENTIFICATION DIVISION

1. GENERAL

The purpose of the Identification Division is to identify or label the Source Program and to provide any other pertinent information concerning the program. The Identification Division is essentially an extended note and the information contained therein becomes a part of the program listing but has no effect on the object program.

The format for the Identification Division is as follows:

IDENTIFICATION DIVISION.

PROGRAM-ID. program-name.

[AUTHOR. author-name.]

[INSTALLATION. any sentence or group of sentences.]

[DATE-WRITTEN. any sentence or group of sentences.]

[DATE-COMPILED. any sentence or group of sentences.]

[SECURITY. any sentence or group of sentences.]

[REMARKS. any sentence or group of sentences.]

2. ORGANIZATION

Fixed paragraph-names are used throughout the Identification Division. As the format indicates, only the PROGRAM-ID paragraph is required; the other paragraphs are optional and the programmer can specify any or all of them.

The PROGRAM-ID paragraph must always appear as the first paragraph of the Identification Division. The program-name designated is a word and must conform to the rule for a word (either a name or a literal). The program-name is always used to identify the source program, the object program and all associated documentation. The paragraph-name DATE-COMPILED will cause the current date and a period to replace the contents of the paragraph.

3. EXAMPLE

The Identification Division of a typical program might be written as follows:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COST-OF-SALES.
AUTHOR. J. A. HOLMES.
DATE-WRITTEN. DEC. 6 1961
DATE-COMPILED. FEB. 15 1962.
REMARKS. FILES NAMED ERROR LISTING AND LOSS-
ITEMS ARE TO BE PRINTED USING
1ST 2 DIGITS OF RECORD AS
PAGE CONTROL.
```

X SPECIAL FEATURES

1. LIBRARIES

a. General

The UNIVAC 1107 standard COBOL library may contain two types of entries, corresponding to the DATA and PROCEDURES divisions of the COBOL system. Each of these divisions is capable of withdrawing from the library only that material pertaining to itself. The two types of entries in the library are:

1) Those entries associated with the DATA DIVISION. These entries consist of File and Record descriptions. This information is retrievable through the use of COPY in the File and Record Description entries of the Data Division.

2) Those entries associated with the PROCEDURE DIVISION. These entries consist of Procedure paragraphs only. This information is retrievable through the use of the COPY statement in paragraph entries in the PROCEDURE DIVISION.

The calling of material from the library described above produces the same effect as if the programmer had written the library material in his source program.

b. The DATA DIVISION Entries in the Library

The entries in the Library associated with the DATA DIVISION are of two types. First, there are the entries relating to the File Description portion of the DATA DIVISION. These are retrieved by the use of the COPY option in an entry carrying an FD level number. Second, there are the entries relating to the Record Description, Common-Storage, Working Storage or Constant Sections

of the DATA DIVISION. These are retrieved through the use of the COPY option at any level in the source program Record Descriptions.

The level number FD must be associated with the first entry of each File Description paragraph in the Library. This will automatically separate the information from the File Descriptions preceding it. The information copied from the library will terminate when another FD level number is encountered in the Library.

In the other portion of the DATA DIVISION, since a COPY will include all lower level entries following the level to be copied, the first entry in each set of the Record Descriptions placed on the library must carry a level number of one (01). This will automatically separate this set from the descriptions which precede it on the Library. This does not restrict the level at which the descriptions may be copied, since the level number of the descriptions being copied is automatically replaced by the level number of the entry in the source program in which the COPY option was used. All other level numbers associated with information copied from the Library are automatically adjusted.

c. The PROCEDURE DIVISION Entries in the Library

A routine (PROCEDURE paragraph) is a COBOL procedure which has been stored in the Library. The routine is composed of one COBOL paragraph. (See COPY statement in PROCEDURE DIVISION, Section VII.) The routine is identified by a procedure-name (i.e., the name of the paragraph).

Routines are retrieved from the Library during compilation through the use of the COPY statement in the PROCEDURE DIVISION. The result of the compilation is the same as if the routine were actually written as part of the source program, but with the name of the source paragraph containing the COPY statement substituted for the Library name of the procedure.

Procedure-names of paragraphs stored in the Library must be unique within the Library.

Procedures for placing information on the Library and other information about Library maintenance will be specified later.

2. SORTING IN UNIVAC 1107 COBOL

The SORT function is a very important function that must exist in a software package in order to facilitate a workable Systems Design. While the function of Sorting is easily understood, the implementation of the function on a computer is not as well understood. To insure success in fulfilling data organization problems and to produce reasonable data processing systems, the COBOL programmer must carefully observe the rules stated here and carefully apply the function.

Input-Procedures

Procedures may easily be inserted between the input subroutine and the beginning of the SORT function. The size of these inserted procedures is always restricted by the size of the first portion of the SORT procedures and data storage. The efficiency of the first portion of the SORT procedure may also be reduced by the size of the procedures which are inserted.

For the COBOL programmer, the UNIVAC 1107 SORT feature consists of subroutines which do not include input and output subroutines of the main program though the SORT does have input-output subroutines to handle the data which has been "given" to the SORT.

In the simplest case, the input procedures the UNIVAC 1107 COBOL programmer must write consist of steps to:

1. Open the input file and the SORT file.
2. Read the input file.
3. Write the SORT file.
4. Close the input and SORT files.

When these procedures are completed the first subroutine associated with the SORT subroutines terminates its functions, and subsequent SORT routines will begin to merge sorted data into a single ordered file. It is important to realize that the SORT file referenced in the steps above does not have the same physical characteristics of other files described by the COBOL programmer. While the method for describing the SORT file is identical to the method used for all other files, the SORT file seldom exists as a contiguous ordered string of records on or in a single storage device. The COBOL programmer is encouraged to read separate documents on the techniques of computer SORT implementation, keeping in mind that during this first pass the detailed operation cycle is approximately:

1. Open Input and SORT files.

2. Read input file record (when the end of the Input file is reached go to step six).

3. "Write" SORT file record.

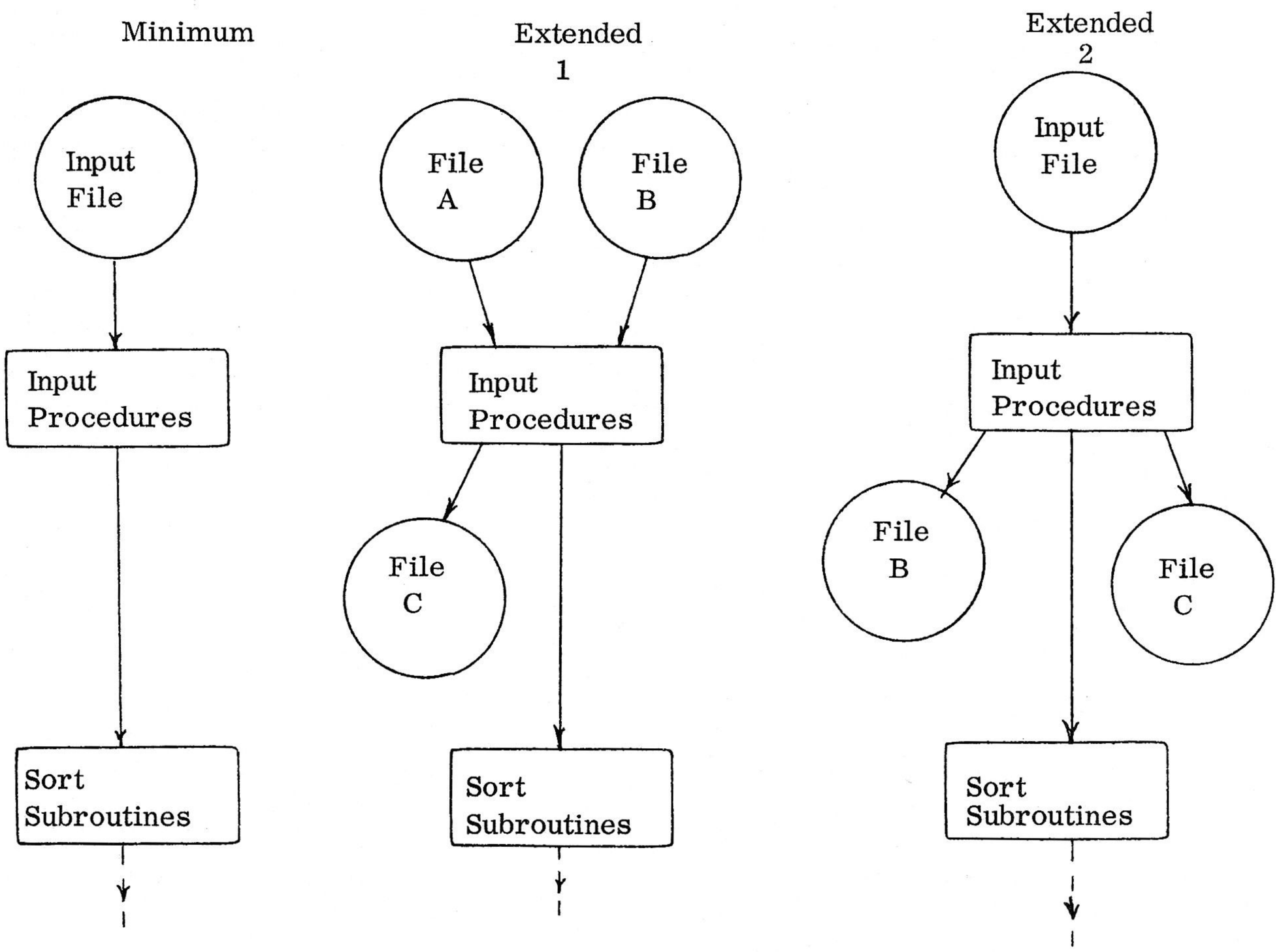
4. Perform the first SORT subroutine processes.

5. Go to step two.

6. Close the input and SORT files.

Step 4 is never stated in the COBOL Source Program because the process is implied. This permits the greatest amount of generalization, allowing the compiler writer to provide any procedure he may elect which best serves the hardware and the sorting function.

The UNIVAC 1107 SORT function may be thought of as more or less an extension to the input/output library, with additional operations inserted, between the READ Input File and WRITE SORT file statements, with no difficulty. Since the SORT subroutines for the UNIVAC 1107 are moderate in their space requirements, it may be desirable to make more complex additions to the SORT Input Procedures which might, as an example, include more than one input file and additional output files. The Input Procedures to a UNIVAC 1107 SORT can be illustrated then as follows:



In the first EXTENDED illustrations File C is an output file of the Input Procedures. Each additional Input Procedure file will require storage for buffering, file and record handling procedures and additional logic to control the multiple Input/Output logic. Some maximum tape to tape computer program size will undoubtedly be established for each installation and simple addition will normally give a good indication of available core storage. With the experience gained from using the compiler, the number of additional procedure statements which may be safely added will be established, and the effect of the additional volume of procedures to the total program time will become an established factor. Data carry over to the Output Procedures will also require storage. In the normal case data carry over is not required, but may be accomplished in UNIVAC 1107 COBOL implementation. See Common Storage.

Output Procedures

Procedures may be easily inserted between the last phase of the SORT subroutines and the output procedures. The size of the object program of the inserted procedures is also restricted by the size of the object program of the last phase of the SORT subroutines and storage. Generally, the Output Procedures one may insert can be considerably larger than the inserted Input Procedures.

In the simplest case the UNIVAC 1107 COBOL programmer must write an Output Procedure consisting of the steps:

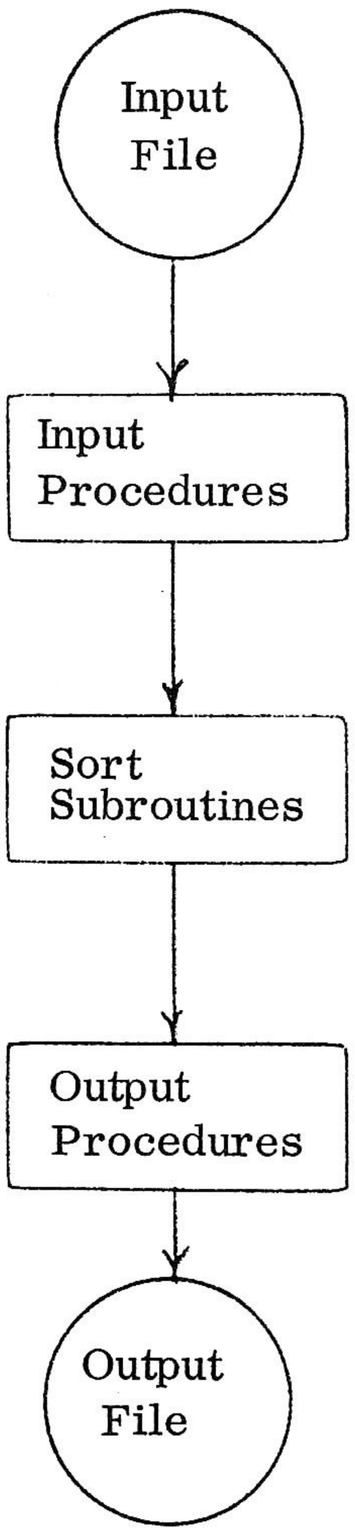
1. Open the SORT file and the Output file.
2. Read the SORT file.
3. Write the Output file.
4. Close the SORT and Output files.

The procedures associated with the last phase of the SORT subroutines (final merge operation) produce sequenced records which are obtained by the UNIVAC 1107 COBOL programmer by READing the SORT file. The detailed operation cycle that must be supplied is approximately:

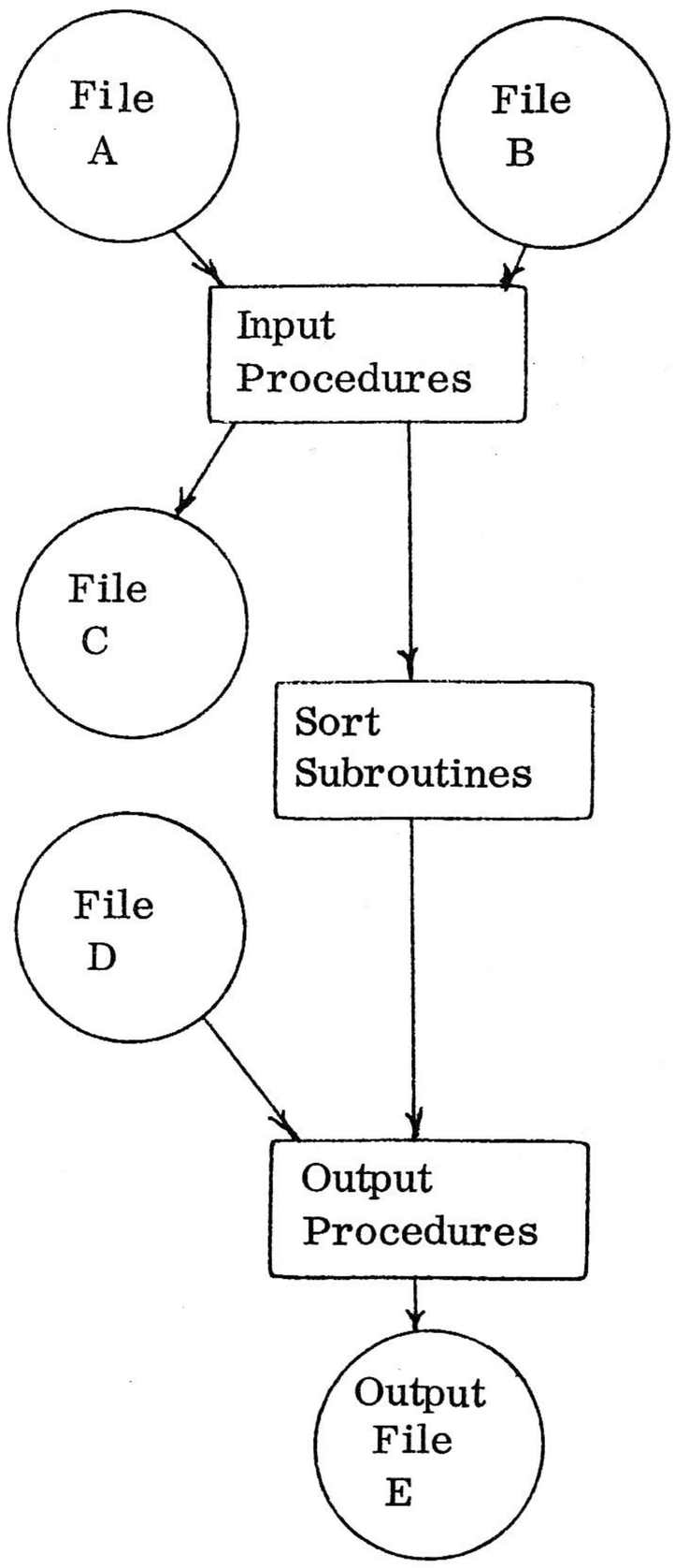
1. Open SORT and output files.
2. Read SORT file. When the end of the SORT file is reached go to step 5.
3. Write Output file.
4. Go to step 2.
5. Close the SORT file and the Output file.

The implied operations of the output sequence are initiated independently of the foregoing steps. Just as in the Input Procedures, additional procedures are easily inserted between the READ of the SORT file and the WRITE of the Output file. The illustration for the Input Procedures can now be extended to:

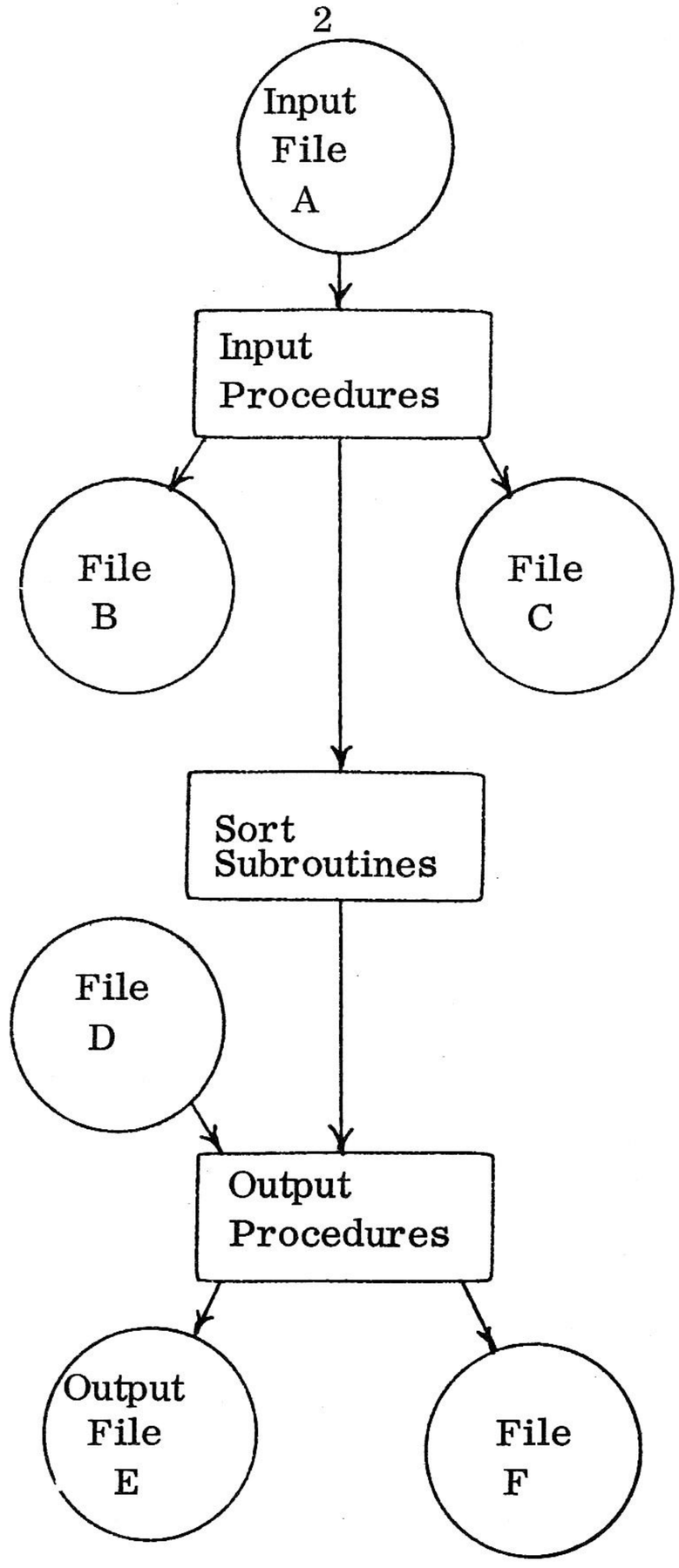
Minimum



Extended 1



Extended 2



It should be noted that the number of input and output files used throughout the SORT procedures has no upper bounds except those imposed by memory capacity and magnetic tape unit availability. It should also be noted that inputs and outputs to Input Procedures as well as to Output Procedures could be parts of OPEN-READ-WRITE-CLOSE sequences associated with other SORT files in the same COBOL Source Program. While this allows great flexibility, it also allows great complications. The user should beware of the consequences.

Programmer Considerations

To produce a COBOL Source Program which will cause a file to be sorted, the UNIVAC 1107 COBOL programmer must provide the following information.

Environment Division

In the FILE-CONTROL paragraph of the INPUT/OUTPUT SECTION the optional word SORT-FILE must be included in the SELECT statement as follows:

FILE-CONTROL. SELECT SORT-FILE
file-name-1 ....

where file-name-1 is the name assigned in the DATA DIVISION to an FD entry which is not directly referenced by any READ or WRITE, OPEN or CLOSE statements in the PROCEDURE DIVISION for other than the SORT-FILE. In the simplest case, the Input file may be renamed in the FILE-CONTROL entry to establish the SORT-FILE description and the Output file description. Alternatively the DATA DIVISION could COPY the FD and record descriptions of the Input file (from the Library) to establish the file descriptions of the SORT-FILE and Output file.

In addition, the number of hardware units must be ASSIGNed in the same clause. Examples of the FILE-CONTROL entry are:

Example 1.

FILE-CONTROL. SELECT MASTER-FILE ASSIGN TO 2 UNISERVO-IIIS
SELECT SORT-FILE MASTER-FILE-SD RENAMING MASTER-FILE
ASSIGN TO 6 UNISERVO-IIIS. SELECT UPDATED-MASTER RENAMING
MASTER-FILE ASSIGN TO 1 UNISERVO-III.

In this example a single FD entry in the DATA DIVISION provides all file descriptions. The programmer should remember that each WRITE record-name statement must use qualification, i.e., WRITE TRANSACTION IN MASTER-FILE-SD.

Example 2.

```
FILE-CONTROL.  SELECT MASTER-FILE ASSIGN TO 1 UNISERVO-III.
    SELECT CHANGE-FILE ASSIGN TO 1 UNISERVO-III.  SELECT
    SORT-FILE ACTIVITY-FILE ASSIGN TO 4 UNISERVO-IIIS.  SELECT
    UPDATED-MASTER ASSIGN TO 1 UNISERVO-III.  SELECT SORTED-
    CHANGES RENAMING CHANGE-FILE ASSIGN TO 1 UNISERVO-III.
    SELECT AUDIT-ERRORS ASSIGN TO 1 UNISERVO-III.
```

In this example the Input Procedures could include an input, Updated-Master, and the Output Procedures could include two outputs, or any logical combination depending upon the PROCEDURE DIVISION.

Other FILE-CONTROL clauses, i.e., FOR MULTIPLE REEL, and RESERVE AREAS, may be used where applicable to Input Files and Output Files of First Pass COBOL procedures and Last Pass COBOL procedures not involving the SORT-FILE. SORT-FILE control is a completely independent system for which these clauses have no meaning.

Data Division

In the FD entry for a SORT-FILE the programmer must specify the sort keys by adding the SEQUENCED ON clause which permits the selection of ASCENDING or DESCENDING following each data-name. It is possible to sort a file on one sequence on the major key and another sequence on a minor key, i.e., to sort a file of Accounts Receivable in Ascending sequence on Customer-name and on Descending sequence on payment dates to order a file so that the data may be presented with the most recent payment first.

It should be noted that sort keys specified in the SEQUENCED clause must always be present in the record. If there is more than one type of record in the file, the keys will always appear in the same physical location in each record written on a SORT-FILE. If they are not, alignment of keys may be accomplished in the First Pass COBOL procedure. If the keys must be realigned, this can be accomplished in the Last Pass COBOL procedure.

Since the label records used by the SORT subroutines are not necessarily identical in nature to STANDARD label records, the LABEL RECORDS ARE clause and the VALUE OF ID clause are not meaningful in the SORT-FILE FD entry; their presence does not constitute an error. They will simply be ignored.

SORT Summary

1. Open the SORT file and the Input file.

2. Each record to be sorted must be READ or generated from input, then be "written" on the SORT-FILE.

   When the First Pass COBOL procedures determines that First Pass COBOL procedures input is exhausted, control in the object program is transferred to step 3. In no case should First Pass COBOL procedures attempt to READ the SORT-FILE. Summarization, deletion, and generation of records to be sorted may be accomplished in First Pass COBOL procedures via:

   a. Normal summary procedures.

   b. READing more than one input record prior to a WRITE on the SORT-FILE.

   c. Writing more than one SORT-FILE output record prior to a READ of input.

3. The SORT-FILE, as well as all other files used in First Pass COBOL procedures, is CLOSEd.

   None of the following procedures may reference the above listed procedures of the First Pass COBOL procedures.

4. The SORT-FILE is OPENed as an input. Any other Last Pass COBOL procedures input files and/or output files are also OPENed.

5. The SORT-FILE is READ.

6. An output file is written upon and control returned directly or indirectly to the SORT-FILE READ, step 5.

7. When Last Pass COBOL procedures determine that Last Pass COBOL procedures input is exhausted, control in the object program is transferred to CLOSE procedures.

An example of the UNIVAC 1107 COBOL procedures for a simple sort is:

```
FPCP SECTION.
START-SORT
     OPEN INPUT input-file-name.
     OPEN OUTPUT sort-file-name.
FP-INPUT-CONTROL
     READ input-file-name RECORD AT END GO TO FPCP-END.
FP-OUTPUT-CONTROL.
     WRITE sort-file-record FROM input-file-record.
     GO TO FP-INPUT-CONTROL.
FPCP-END.
     CLOSE input-file-name, sort-file-name.
LPCP SECTION.
FINISH-SORT.
     OPEN INPUT sort-file-name.
     OPEN OUTPUT output-file-name.
LP-INPUT-CONTROL
     READ sort-file-name RECORD AT END
     GO TO END-OF-JOB.
LP-OUTPUT-CONTROL.
     WRITE output-file-record FROM sort-file-record.
     GO TO LP-INPUT-CONTROL.
END-OF-JOB.
     CLOSE sort-file-name, output-file-name.
     STOP RUN.
```

3. COMPILER DIAGNOSTICS

Whenever a source COBOL program is compiled programming errors may appear and where a programmer has made errors some notation of these must be noted as aids to program corrections to be made by the programmer. As an output of the compilation a listing is made of the original source program with marks or remarks alongside of the error lines to indicate what sort of error was made. A complete list of such marks and remarks will appear later.

These diagnostics will be divided into four kinds each of which will be treated in a different manner:

1. Precautionary diagnostic - a precautionary diagnostic will be produced when the statement is a potential source of error. In effect the compiler is saying, "We did what you requested, but did you really mean this? Check your program to see if other statements agree."

2. Correctible error. If the statement contains an error which is believed to be correctible or which can be processed correctly but is illegal according to COBOL rules then the correction will be attempted and a diagnostic message produced.

   In effect the compiler is saying, "COBOL rules have been violated and I have second-guessed your logic."

3. Uncorrectible error. If a statement or clause contains an error which is not "correctible" then the statement or clause will be rejected. In effect the compiler is saying, " I haven't enough information to process this statement and I cannot second-guess your logic. Rewrite this and submit it to me again."

4. Destructive errors. If an error is such as to destroy the usefulness of the compilation then the compilation will be terminated at the place where it is expected that no more useful diagnostic information is obtainable.

   In effect the COBOL compiler is saying, "The program has come to a point where your previous errors have so multiplied that the logic I am trying to keep track of is completely confused. It is useless to go further. Rewrite the program."

4. MONITOR VERB

The verb MONITOR is used in a procedure paragraph referencing certain data to check and see if the procedure behaves in the manner in which the programmer thought it would. It is an object program check, effective and active only on the object program actually running on the computer.

The MONITOR function is only effective in the clause where it is written and does not apply to the complete program. Also it only checks the data the programmer referenced in the MONITOR statement.

This verb would be used in program testing a newly compiled program using test data or in trying to find an error as evidenced in the output of a program. See MONITOR verb in PROCEDURE DIVISION, Section VIII.

5. SEGMENTATION

A source program is written in sections each of which normally represents a series of closely related operations involved in performing a particular function. After all sections have been completed, they can be classified according to their importance or frequency of use and their relation to other sections. This classification is accomplished by means of a priority which is written after the SECTION name as follows:

Section-name SECTION priority.

The most frequently used Sections will have a priority of zero (0). All sections with the same priority will operate as one segment. Therefore a segment is a section or sections which has an established fixed relationship. When one portion (section) of á segment is in memory, all other portions of that same segment must also be in memory. Sections which do not operate together closely or frequently should not have the same priority number unless they all are zero.

The compiler assumes that the logical sequence of the program is the same as the physical order except for programmed transfer of control; in other words, a sequential order of instructions.

Segmentation allows the less frequently used routines to be outside of memory except when needed. Two or more outside routines can share the same area alternately, provided they do not communicate from one to the other while in memory.

Segments are handled in two ways according to their priority:

1. SECTIONs having a priority of 1 to (but not including) 50 are included in the "zero segment" of the object program. The zero segment of the program is the portion of the program which must be in memory at all times. If sufficient memory is not available for the zero segment the compilation will continue, a diagnostic warning will be given and the results of the object program will be unpredictable.

2. SECTIONs having priorities from 50 through 99 will be grouped by priority number into segments. The memory area "set aside" for containing these segments alternately (not in priority sequence but as called) will be equal in size to the largest segment.

SECTIONs which are not included in the zero segment are grouped into segments by priority. The rules for handling these segments are:

1. Each segment must be able to be called in without disturbing the zero segment memory area requirement.

2. The compiler will provide for locating the "outside" segment, transferring it to the memory set aside for it, establish all communication requirements to permit cross-talk to the zero segment.

3. A GO statement in a section whose priority is <u>greater</u> than 49 cannot be referenced by an ALTER in a section having a <u>different</u> priority. Control can be transferred to any point within a segment; it is not restricted to the starting point.

The programmer does not issue calls of any type to bring the segments into memory. A reference, in the zero segment, to any "outside" segment will cause the compiler to generate the call. The segment referenced will be brought in before continuing with the program. The programmer therefore should exert due care not to make promiscuous references to "outside" segments.

The following points should be considered when making priority assignments:

1. The assignment of priority numbers or the determination of a need for them is a programmer responsibility.

2. Proper use of segmentation for greatest program efficiency is up to the programmer. The compiler has no means of determining either need for or the use of segmentation.

3. If all of a program can be in memory at one time, there is no need for segmentation.

4. Segmentation is a very useful feature but it must be used properly. The normal practice is to keep "outside" segments to a minimum and then only infrequently used functions, e.g., initialization and/or close-out. In the extreme case, however, segmentation could be used to almost completely overlay one program with another by assigning only a small control function as priority zero and assigning the remaining "major" program functions different priority numbers.

5. Space in memory will be preserved equal to the size of the largest overlay segment regardless of how many there may be.

6. One and only one priority overlay segment can be in memory at one time.

The following diagrams are some examples of segmentation use of memory.

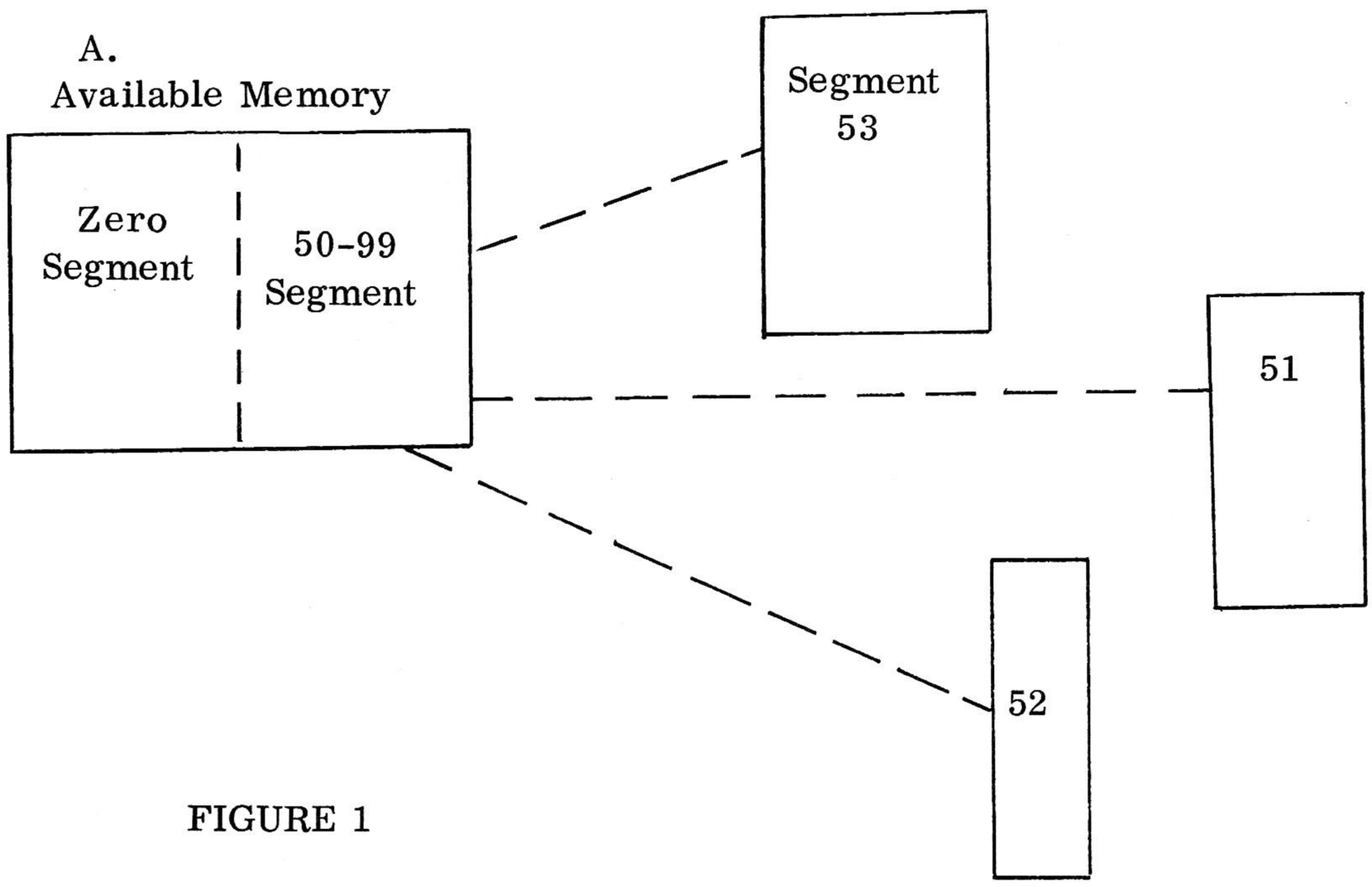


FIGURE 1

Segments 51, 52 and 53 in Figure 1 represent the size of these segments and do reflect where they are or in what order they appear on tape. "Available Memory" means memory allocated for this program not necessarily all the computer. Each segment could be called in as often as required by the zero segment, and they would overlay one another in memory.

B.

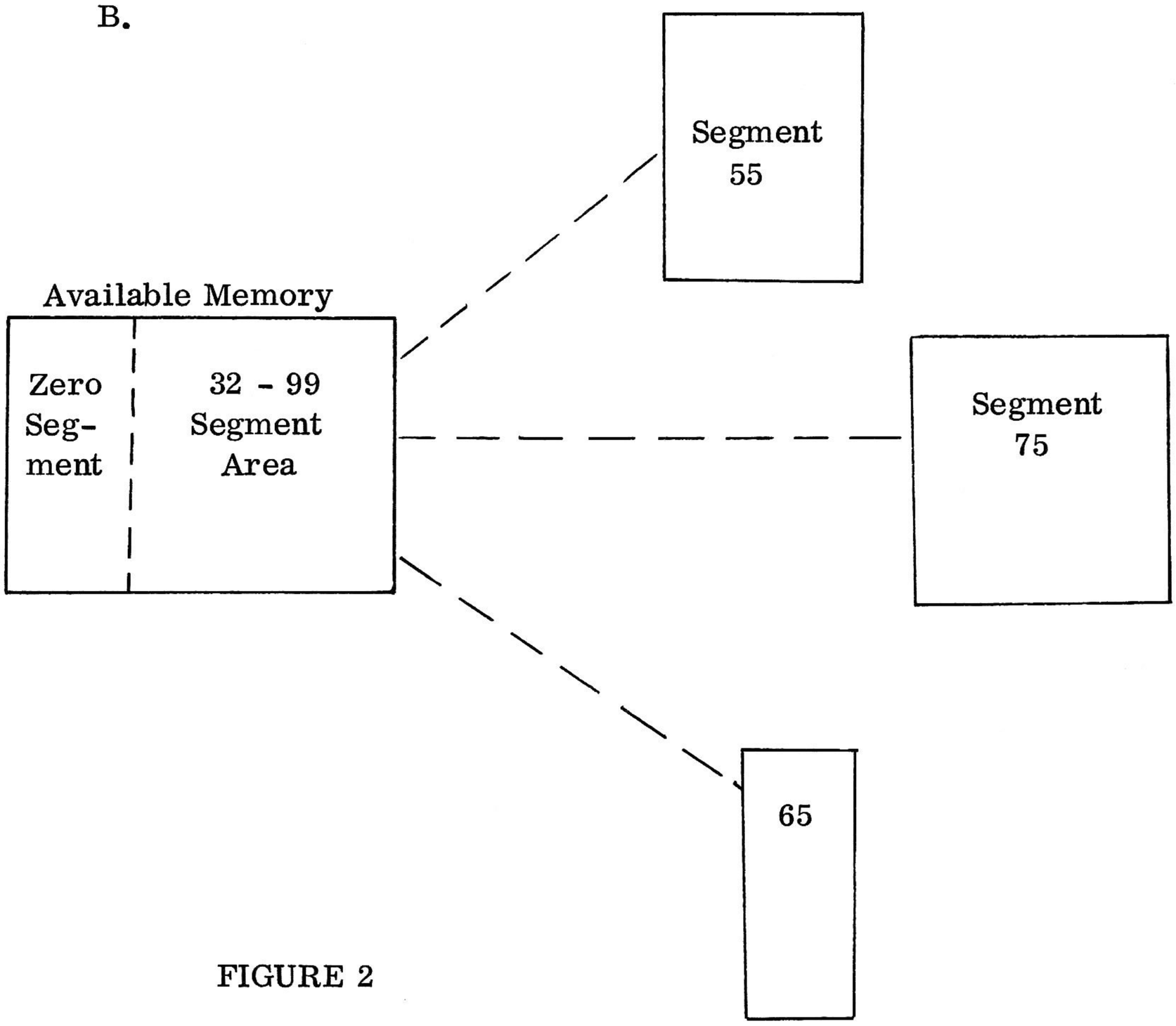


FIGURE 2

Figure 2 is another example to indicate that the size of segments 32-99 need not be equal or in any required order. Note, segment 75 is equal in size to 32-99 segment in available memory.

6. INDEPENDENTLY COMPILED SUBPROGRAMS

The UNIVAC 1107 COBOL compiler has been designed to permit a maximum of flexibility of system design. An excellent example of this is the ability to compile subprograms independently and thereby derive the many benefits of smaller, more manageable compilations. For example, changes can be incorporated if they affect only one subprogram by reprogramming and compiling that subprogram only.

The means of communication between independently compiled subprograms and the main program are provided by the ENTER and RETURN verbs and the FILE and COMMON-STORAGE sections of the Data Division.

Any given COBOL program may be subdivided into two or more parts, each of which can be compiled independently.

Each subprogram, as well as the main program, is similar to a complete, simple COBOL program. All must have:

an IDENTIFICATION DIVISION
an ENVIRONMENT DIVISION
a DATA DIVISION
a PROCEDURE DIVISION

Successful usage of subprograms is subject to certain rules.

1. The means of communication between the Procedure Divisions of a main program and its subprograms is through the use of ENTER and RETURN statements.

2. The FILE SECTION of each program must contain File and Record description entries for all files referenced in that program.

Each program will have its own separate WORKING STORAGE Section and CONSTANT Section. Data described in these sections cannot be shared between one program and another.

Memory will be allocated for the files described in the FILE SECTION of any program. All subprograms making reference to files or file data will, when incorporated with other programs at run time, reference a single area of memory allocated for each of the files referenced. Memory will be allocated for the data described in the COMMON-STORAGE SECTION of the last program compiled and placed in the package of programs. All subprograms making reference to COMMON-STORAGE data will, when incorporated with the other programs at run time, reference that area of memory allocated for the COMMON-STORAGE SECTION for all the programs. Therefore, at run time there will be only one memory area reserved for file data and only one memory area reserved for COMMON-STORAGE data and these areas will be automatically shared by both the main program and all of its subprograms.

The Sample Problem Listing referred to throughout the Guide will be supplied at a later date.

COBOL

UNIVAC

DIVISION OF SPERRY RAND CORPORATION

PRINTED IN U.S.A. U-2582