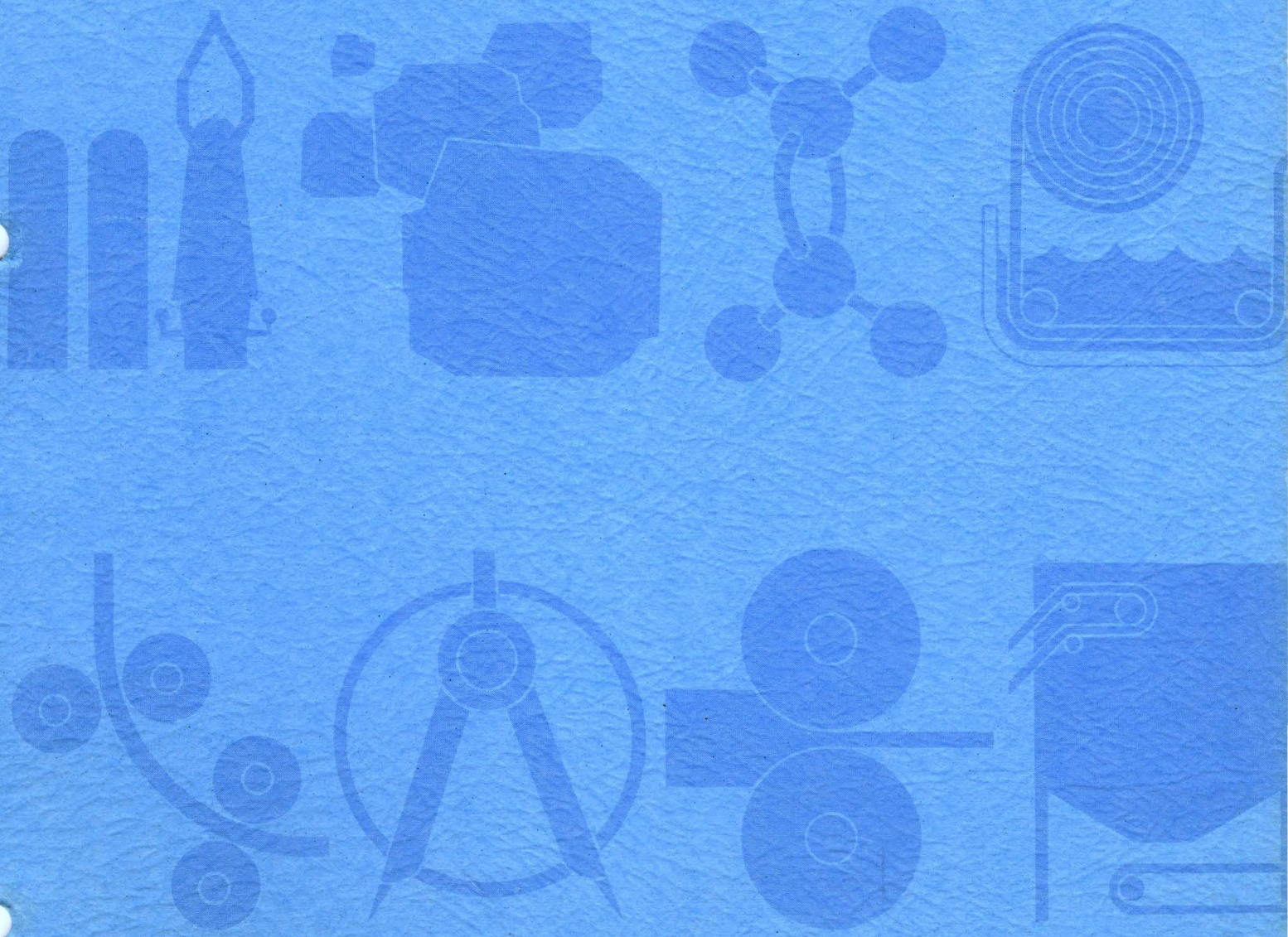
Honeywell

COBOL REFERENCE MANUAL

LEVEL 66
SOFTWARE



USS Engineers and Consultants, Inc.
a Subsidiary of United States Steel Corporation

Honeywell

COBOL REFERENCE MANUAL ADDENDUM A

SERIES 60 (LEVEL 66)/6000

SOFTWARE

SUBJECT:

Additions and Changes to the Series 60 (Level 66)/6000 COBOL Reference Manual.

SPECIAL INSTRUCTIONS:

This update, Order Number DD25A, is the first addendum to DD25, Rev. 0, dated June 1975. The attached pages are to be inserted into the manual as indicated in the collating instructions on the back of this cover. Change bars in the page margins indicate technical additions and changes; asterisks indicate deleted material. These changes will be incorporated into the next revision of the manual.

For this software release, the syntax construct 6000 WITH EIS (SOURCE-COMPUTER and OBJECT-COMPUTER paragraphs) may also be specified as 6000-EIS. The USAGE COMP-3 PACKED SYNC construct may also be specified as USAGE COMP-4. This release also includes a new OPTIMIZE COMPUTATIONAL option in the SPECIAL-NAMES paragraph.

NOTE: This cover should be inserted following the manual cover to indicate that the document has been updated with Addendum A.

SOFTWARE SUPPORTED:

Series 60 Level 66 Software Release 3
Series 6000 Software Release I

DATE:

February 1977

ORDER NUMBER:

DD25A, Rev. 0

20445
2478
Printed in U.S.A.

COLLATING INSTRUCTIONS

To update this manual, remove old pages and insert new pages as follows:

Remove	Insert
3-1, 3-2	3-1, 3-2
3-5, 3-6	3-5, 3-6
3-9, 3-10	3-9, 3-10
3-17, 3-18	3-17, 3-18
3-21, blank	3-21, blank
5-3 through 5-12	5-3 through 5-10
	5-10.1, blank
	5-11, blank
	5-11.1, 5-12
5-19 through 5-26	5-19 through 5-24
	5-25, blank
	5-25.1, 5-26
5-27, blank	5-27, blank
6-5, 6-6	6-5, 6-6
6-13 through 6-16	6-13 through 6-16
6-31 through 6-34	6-31 through 6-34
6-47, 6-48	6-47, 6-48
6-67, 6-68	6-67, 6-68
6-73 through 6-76	6-73 through 6-76
7-21 through 7-24	7-21 through 7-24
7-27 through 7-30	7-27 through 7-30
7-33, 7-34	7-33, 7-34
7-37 through 7-40	7-37, 7-38
	7-38.1, blank
	7-39, 7-40
7-45, 7-46	7-45, 7-46
7-49, 7-50	7-49, 7-50
7-61 through 7-66	7-61 through 7-66
7-69, 7-70	7-69, 7-70
7-75, 7-76	7-75, 7-76
7-85 through 7-88	7-85 through 7-88
7-93 through 7-96	7-93 through 7-96
7-99, 7-100	7-99, 7-100
7-107, 7-108	7-107, 7-108
	7-109, blank
A-1 through A-4	A-1 through A-4

 File No.: 1723,1P23

SERIES 60 (LEVEL 66)/6000

SOFTWARE

COBOL REFERENCE MANUAL

SUBJECT

Complete Description of COmmon Business Oriented Language (COBOL) Implemented Specifically for the Series 60 (Level 66) and Series 6000 Information Systems

SPECIAL INSTRUCTIONS

This manual replaces COBOL Reference Manual, Order Number BS08, for Series 6000 system users. Order Number BS08 must be used by Series 600 system users and by Series 6000 system users who are on prior software releases.

Those COBOL features implemented for the Series 60 (Level 66) and 6000 systems that are nonstandard or unique to the Series 60 (Level 66)/6000 systems are indicated by shading. In addition, those features of COBOL 1968 presented in this manual that have not been implemented are indicated as such by delta symbols (▲) in the margins of those pages on which these features are presented.

SOFTWARE SUPPORTED

Series 60 (Level 66) Software Release 2
Series 6000 Software Release H

ORDER NUMBER

DD25, Rev. 0

June 1975

Honeywell

PREFACE

This COBOL Reference Manual and a companion manual, the COBOL User's Guide, have been prepared for Series 60 and Series 6000 users.

This manual is organized in a format similar to that used in the Conference on Data Systems Languages (CODASYL) COBOL Journal of Development (JOD). It is intended to be strictly a language reference document containing the formats, syntax rules, general rules, and special considerations required to construct a COBOL source program. The contents of the manual reflect (a) Series 60/6000 COBOL as it relates to the Series 60/6000 operating system; (b) implemented elements of CODASYL COBOL as published in the JOD; and (c) American National Standard COBOL (X3.23-1968). The highest level of American National Standard implementation is presented for all modules, with some minor exceptions indicated by delta (▲) symbols.

As a supplement to the COBOL Reference Manual, the COBOL User's Guide provides information concerning COBOL concepts, Series 60/6000 implementation techniques, internal compiler characteristics, and efficiency considerations. In addition, sample deck setups and job control data are provided to assist the user in interfacing with the operating system.

The following language elements, defined in previous versions of the COBOL specifications, have been deleted from CODASYL COBOL and are not included in American National Standard COBOL; however, they are still supported by Series 60/6000 COBOL in the current state of implementation:

- Figurative constants: HIGH-BOUNDS and LOW-BOUNDS
- PREPARED option
- Constant Section
- CLASS clause
- Editing clauses
- FILE CONTAINS clause
- POINT LOCATION clause
- RANGE clause
- SEQUENCED clause

 File No.: 1723,1P23

- SIGNED clause
- SIZE clause
- Conditional statement IF...OTHERWISE...THEN...

As a continuing policy of conforming to COBOL standards and encouraging program transferability, documentation of the above language elements has been deleted from this manual. For a detailed description of the obsolete language elements, refer to Section XVII of the COBOL User's Guide.

Series 60 Level 66 is hereafter referred to as Series 60. The technical information contained in this manual refers to both the Series 6000 and Series 60 systems, unless otherwise specifically stated.

ACKNOWLEDGMENT

This acknowledgment has been reproduced from the "Journal of Development, 1968" as requested in that publication, prepared and published by the CODASYL COBOL Programming Language Committee.

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

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the COBOL 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.

The authors and copyright holders of the copyrighted material used herein

FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications."

FUNCTIONAL LISTING OF PUBLICATIONS
for
SERIES 60 (LEVEL 66) and SERIES 6000 SYSTEMS

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE Series 60 (Level 66)/Series 6000:	ORDER NO.
Hardware reference:		
Series 60 Level 66 System	Series 60 Level 66 Summary Description	DC64
Series 6000 System	Series 6000 Summary Description	DA48
DATANET 355 Processor	DATANET 355 Systems Manual	BS03
DATANET 6600 Processo	DATANET 6600 Systems Manual	DC88
Operating system:		
Basic Operating System	General Comprehensive Operating Supervisor (GCOS)	DD19
Job Control Language	Control Cards Reference Manual	DD31
Table Definitions	System Tables	DD14
I/O Via MME GEINOS	I/O Programming	DB82
System initialization:		
System Startup	System Startup	DD33
System Operation	System Operation Techniques	DD50
Communications System	GRTS/355 and GRTS/6600 Startup Procedures	DD05
Communications System	NPS Startup	DD51
DSS180 Subsystem Startup	DSS180 Startup	DD34
Data management:		
File System	File Management Supervisor	DD45
Integrated Data Store (I-D-S)	I-D-S/I Programmer's Guide	DC52
Integrated Data Store (I-D-S)	I-D-S/I User's Guide	DC53
File Processing	Indexed Sequential Processor	DD38
File Input/Output	File and Record Control	DD07
File Input/Output	Unified File Access System (UFAS) (Series 60 only)	DC89
I-D-S Data Query System	I-D-S Data Query System Installation	DD47
I-D-S Data Query System	I-D-S Data Query System User's Guide	DD46
Program maintenance:		
Object Program	Source and Object Library Editor	DD06
System Editing	System Library Editor	DD30
Test system:		
Online Test Program	Total Online Test System (TOLTS)	DD39
Test Descriptions	Total Online Test System (TOLTS) Test Pages	DD49
Error Analysis and Logging	Honeywell Error Analysis and Logging System (HEALS)	DD44
Language processors:		
Macro Assembly Language	Macro Assembler Program	DD08
COBOL-68 Language	COBOL	DD25
COBOL-68 Usage	COBOL User's Guide	DD26
JOVIAL Language	JOVIAL	DD23
FORTRAN Language	FORTRAN	DD02
Generators:		
Sorting	Sort/Merge Program	DD09
Merging	Sort/Merge Program	DD09

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE	ORDER NO.
	Series 60 (Level 66)/Series 6000:	
Simulators:		
DATANET 355/6600 Simulation	DATANET 355/6600 Simulator	DD32
Service and utility routines:		
Loader	General Loader	DD10
Utility Programs	Utility	DD12
Utility Programs	UTL2 Utility Routine (Series 60 only)	DC91
Media Conversion	Bulk Media Conversion	DD11
System Accounting	Summary Edit Program	DD24
FORTRAN	FORTRAN Subroutine Libraries	DD20
FNP Loader	DATANET 355/6600 Relocatable Loader	DD35
Service Routines	Service Routines	DD42
Software Debugging	Debug and Trace Routines	DD43
Time Sharing systems:		
Operating System	TSS General Information	DD22
System Programming	TSS Terminal/Batch Interface	DD21
System Programming	TSS System Programmer's Reference Manual	DD17
BASIC Language	Time Sharing BASIC	DD16
FORTRAN Language	FORTRAN	DD02
Text Editing	Time Sharing Text Editor	DD18
Remote communications:		
DATANET 30/305/355/6600 FNP	Remote Terminal Supervisor (GRTS)	DD40
DATANET 355/6600 FNP	Network Processing Supervisor (NPS)	DD48
DATANET 700 RNP	RNP/FNP Interface	DB92
Transaction processing:		
User's Procedures	Transaction Processing System User's Guide	DD41
Handbooks:		
System-operator communication	System Console Messages	DD13
Pocket guides:		
Control Card Formats	Control Cards and Abort Codes	DD04
FORTRAN	FORTRAN Pocket Guide	DD82

CONTENTS

SECTION I

INTRODUCTION

COBOL is an acronym for the phrase COmmon Business Oriented Language. The COBOL system, which includes a compiler (or language processor) in addition to the COBOL language, is used to state all the facets of a business-oriented problem and to convert the statements into a form usable by a computer. The following sections of this manual describe the COBOL language. The COBOL compiler is not described since the purpose of this manual is to give the user an insight into using the language to state the problem most efficiently. A description of the components of the compiler or how the compiler converts the statement of the problem into the ones and zeros a computer uses is not germane to using the COBOL language. This manual, therefore, is for those who understand business-oriented problems and the concepts of data processing.

A business-oriented data processing problem can be broken down into four distinct groups of logically related information. The first is the identification of the problem such as, is it an inventory control problem, a personnel accounting problem, a payroll problem, or a billing problem? Also included in this group is information such as the assignment to solve the problem, when, and where. The second group of logically related information is the data processing environment in which the problem is to be solved. That is, what computer will be used to compile the program and run the job? What peripheral equipment is necessary to run the job? What other programs (or software) are necessary? The information in this group is also useful when a program has been written for one computer environment and it is desired to run it in another computer environment. The third group of logically related information is that in which the data to be processed and the processed data is described. Each file, both input and output, is described in terms of its records. Each unique type of record in a file is described in terms of its unique data items. In addition to this, the organization of the files must be described and the processing mode to be used must be stated. These three groups of logically related information; the identification of the problem, the information related to the data processing environment, and the description of the data, can be considered as the problem statement. The fourth group of logically related information can be considered as the procedure(s) by which the data is to be processed to solve the problem. In this group of information, the user states in a step-by-step manner exactly what is to be done to the data to produce new or additional data.

The COBOL language is structured to accommodate the four groups of logically related information in four named divisions. These divisions are the Identification Division, the Environment Division, the Data Division, and the Procedure Division. Every COBOL program contains these four divisions in the above order. The Identification Division is used to identify by name the source program (that which the user writes) and the outputs of a compilation. Other information that can be included in the Identification Division is the name of the programmer, the name of the installation at which the program was written, the date the program was written, the date of compilation, and any other desired

information such as a brief statement of the purpose of the program. The Environment Division is that part of the program in which the computer(s) to be used for compiling and running the program is described. In the Environment Division, names may be assigned to peripheral equipment and the features of the files directly related to the hardware may be described. In the Data Division, the files of data the program processes or creates and the unique individual records of these files are described. Data is written according to a standard data format rather than an equipment-oriented format. In the Procedure Division, a step-by-step logical process is written to instruct the computer to process the input data.

In the Identification Division, the problem is identified by name, which may or may not reflect the type of problem. For instance, the problem may be named INVCON (for Inventory Control) but the type of inventory control problem may be a combination of updating a master inventory file and producing records from which new parts can be ordered. COBOL, being a business problem oriented language, incorporates features that make it possible to accomplish a particular job without programming the job in detail. The COBOL language allows the user to perform arithmetic calculations, edit data, sort data, merge data, and produce reports. A single COBOL program can perform one or any combination of these functions.

SECTION II

FUNCTIONAL CONCEPTS

GENERAL DESCRIPTION OF COBOL

COBOL is a programming language used throughout the world for programming business data processing applications. The COBOL language was developed by a group of computer users and manufacturers, and first documentation was distributed in April, 1960. Since then it has undergone many changes and extensions resulting from manufacturer experience with COBOL implementation and user experience with COBOL programming for computers of many sizes and configurations. The improvements are embodied in this version of the language termed COBOL-68.

COBOL allows computers to be programmed in a language that is similar to the English language. English paragraphs, sentences, and phrases are written, following the conventions of a standard reference format, to describe the data to be processed and to specify the required procedures. The resulting text is called a COBOL 'source program'.

The source program text consists of lines containing a maximum of 80 characters and is often keypunched on 80-column cards. The source program is submitted as input to the computer under the control of a special program known as a compiler. As output, the compiler produces an object program on punched cards, magnetic tape, or other suitable storage media. The object program is the actual sequence of machine instructions required to accomplish the functions specified in the source program. In addition, the compiler produces an edited listing, which includes an annotated printout of the source program in the reference format. Another important function of the compiler is to analyze the source program for correct COBOL syntax, and to print error comments for any syntax errors that are detected. The computer's operation under control of the compiler is called compilation.

COBOL FUNCTIONAL CONCEPTS

The Procedure Division in COBOL corresponds to the overall program in some other programming languages. In COBOL, however, the Data Division also plays a central role. Procedural statements are formed by combining COBOL reserved words, literals, and data-names.

The COBOL object program typically processes one or more files of data records, and the user exercises considerable control over the actual physical format of each record. A record can contain a few or many individual items of data; the respective items within a record may have quite different formats, and may be related to each other in complex ways. A file can contain several distinct record types.

In the File Section of the Data Division, the user provides a description of each file, including:

- The types of data records in the file.
- The various data items of which each record is composed.
- The detailed format of each item.

The user assigns a name (data-name) to each file, record, and data item, to permit references elsewhere in the source program.

The concept of records is extended to working-storage data items that do not belong to files. Records in working-storage are described in exactly the same manner as records belonging to files, and can have equal complexity. It is also possible to describe independent Working-Storage Section data items that are not structured into records.

In the Procedure Division, many different types of statements may be utilized, to accomplish such functions as the following:

- Reading a record from an input file.
- Writing a record on an output file.
- Moving data to or from the current record of an input or output file, or to or from a working-storage data item. The unit of data moved can be the entire contents of a record or the contents of a data item within a record.
- Using the values of various items, calculating a new value arithmetically through addition, subtraction, multiplication, or division, and storing the new value in a specified item.
- Comparing the values of data items, and executing alternative sets of statements depending upon the outcome of the comparison.
- Transferring control, so that execution continues in another part of the program. A transfer of control may include provision for control to be returned when a specified point is reached, or the return provision may be omitted.
- Generating a line (or set of lines) for a report.
- Sorting or merging a collection of records. Procedural statements may present the records one by one as sorting or merging takes place or an entire file may be submitted, with input housekeeping implicitly provided. Similarly, an output file may be produced implicitly, or procedural statements may receive the records one by one as their final order is established.

- Obtaining special information, such as current date and time of day, from GCOS; or accomplishing low-volume data transmission.
- Beginning and concluding processing of each file.
- Beginning and concluding presentation of each report.
- Setting procedural switches; in effect, changing the destinations of GO TO statements during execution of the object program.
- Accepting an input transaction from a remote device (terminal).
- Displaying Transaction Processing System messages on one or more remote devices.
- Manipulating an item character by character, changing or counting certain character values.
- Concluding execution of the object program.
- Causing user-supplied procedures to receive control during the implementation of certain otherwise automatic input-output and reporting functions.
- Enhancing source program documentation with explanatory notes.
- Accomplishing special functions that are not defined in the COBOL language specifications but are available in other languages. For example, transferring control to another separately compiled or assembled program, or executing an arbitrary sequence of GMAP instructions.
- Explicitly defining control exit points, enhancing programming convenience.

In addition to the preceding Procedure Division functions, COBOL contains sorting, merging, report writing, table handling, mass storage, and segmentation features.

RECORD ORDERING

The ability to arrange records into a particular order or sequence is a common requirement of a data processing user. The sort and merge features of COBOL provide facilities to meet this requirement.

While both of these features are concerned with record ordering, the functions and capabilities of the SORT and MERGE statements are different in a number of respects. The sort will produce an ordered file from one or more files that may be completely unordered in the sort sequence. The merge, however, can only produce an ordered file from two or more files, each of which is already ordered in the merge sequence. The sort is designed to allow for unordered input. The merge is designed to operate upon ordered files with merging accomplished by record comparison as records are encountered during the reading of all files. Only the SORT statement is provided with an optional input procedure, but both the SORT and MERGE statements are provided with optional output procedures.

Sorting

Sorting constitutes a significant percentage of the workload in a business data processing operation. Therefore, an efficient sort program is required in any business software system.

In many sort functions, it is necessary to apply special processing to the contents of a sort file. This processing may consist of adding, deleting, creating, altering, editing, or other modification of the individual records in the file. It may be necessary to apply the special processing before or after the records are reordered by the sort, or the processing may possibly be required in both cases. The COBOL sort feature allows the user to express these procedures in the COBOL language and to specify at which point, before or after the sort, they are to be executed. A COBOL program may contain any number of sorts, and each of them may have independent special procedures. The sort feature automatically causes these procedures to be executed at the specified point in such a way that extra passes over the sort file are not required.

The normal organization of a COBOL program containing a sort is such that the input file is read and operated upon by an input procedure. Within this input procedure, the RELEASE statement is used to create the sort file. That is, at the conclusion of the input procedure, the sort file is composed of those records that have been output by using the RELEASE statement rather than the WRITE statement, and this file is available only to the SORT statement. The execution of the SORT statement arranges the entire set of records in the sort file according to the keys specified in the SORT statement. The sorted records are made available from the sort file using the RETURN statement during the output procedure.

The sort file has no label procedures which the user can control; the rules for blocking and for allocation of internal storage are peculiar to the SORT statement. The RELEASE and RETURN statements imply nothing with respect to buffer areas, blocks, or reels. A sort file, then, may be considered as an internal file which is created from the input file (RELEASE), processed (SORT), and then made available to the output file (RETURN). The sort file itself is referred to and accessed only by the SORT statement.

For additional information concerning the sort process, refer to Section IX of the COBOL User's Guide.

Merging

Merging is a useful technique for combining the contents of two or more files. In some applications, it is necessary to apply special processing to the contents of a merged file. This processing may consist of adding, deleting, altering, editing, or other modification of the individual records in the file. The COBOL merge feature allows the user to express an output procedure in COBOL language to be executed as the merged output is created. The merged records are made available from the merge file by using the RETURN statement in the output procedure.

The merge file has no label procedures which the user can control and the rules for blocking and for allocation of internal storage are peculiar to the MERGE statement. The RETURN statement implies nothing with respect to buffer areas, blocks, or reels. A merge file, then, may be considered as an internal file which is created from input files by combining them (MERGE) as the file is made available (RETURN) to the output file. The merge file itself is referred to and accessed only by the MERGE statement. A merge file description is considered as a particular type of file description. That is, a merge file, like any file, is a set of records.

For additional information concerning the merge process, refer to Section IX of the COBOL User's Guide.

REPORT WRITING

The production of reports has always placed a heavy burden in terms of machine time and programmer time on the business data processing user. The Report Writer feature is available to specify and produce reports quickly and accurately in COBOL. The Report Writer allows the user to describe reports pictorially in the Data Division, thereby minimizing the amount of Procedure Division coding necessary. In the Report Writer feature, the physical aspects of the report format must be distinguished from the conceptual characteristics of the data in the report.

When describing the physical aspects of the report, consideration must be given to the hardware device on which the report is to be written and to the structure and format of the individual page. Facilities for specifying this information are included in the Report Writer entries.

The concept of a hierarchy of levels is used in defining the logical organization of the report. Each report is divided into report groups which in turn are divided into sequences of items. The use of a level structure permits the user to refer to an entire report-name, major or minor report groups, elementary items within report groups, etc.

In creating the report, the user must define necessary report groups. A report group may be of any of the following; HEADING group, FOOTING group, CONTROL group, or DETAIL PRINT group. A report group may extend over several actual lines on the page.

The report description entry contains information pertinent to the overall format of the named report and uses the level indicator RD. The characteristics of the report page are outlined by describing the number of physical lines per page and specifying the limits for presentation of headings, footings, and detail lines. Data items that act as format controls for a report are specified in the RD entry. Each report associated with an output file must be defined by an RD entry.

A report group is a set of data that is composed of several print lines consisting of many data items or one print line containing only one data item. A report group description entry contains, in addition to other information, a level-number and a TYPE description. The level-number indicates the relative position in the data hierarchy of the report groups, and the TYPE clause describes the purpose of the report group in terms of its presentation within the report.

Specifically, the report group description entry defines the format and characteristics for a report group, whether this group is a line, a series of lines, or an elementary item. The relative placement of items within a report group, the level of a particular report group within the hierarchy of report groups, the format of all items, and any control factors associated with the group are defined in this entry.

Schematically, a report group is a line or a series of lines. The length of a line is determined by the compiler from environmental specifications. Initially, the lines consist of all spaces. Within a report, the order of the individual report groups is not significant. Within a report group, the user describes the elements consecutively from left to right and then from top to bottom. The description of a report group is analogous to the description of a data record except that in the report group spaces are assumed where no specific entry is indicated for presentation, while in the data record every character position must be explicitly defined, regardless of its data content. Report Writing is discussed in more detail in Section VIII of the COBOL User's Guide.

TABLE HANDLING

Tables of data are common components of business data processing problems. Although the items that make up a table could be described as contiguous data items, there are two reasons why this approach is not satisfactory. First, from a documentation standpoint, the underlying homogeneity of the items would not be readily apparent; and second, it would be difficult to make an individual element of such a table available if a decision is required to make one of these elements available at object program execution.

Tables composed of contiguous data items are defined in COBOL by including the OCCURS clause in their data description entries. This clause specifies that the item is to be repeated as many times as stated. The item is considered to be a table element and its name and description apply to each repetition or occurrence. Since each occurrence of a table element does not have a unique data-name assigned to it, reference to a desired occurrence may be made only by specifying the data-name of the table element together with the occurrence number of the desired table element. The occurrence number is known as a subscript, and the technique of specifying individual table elements is called subscripting.

The number of occurrences of a table element may be specified as fixed or variable. If the occurrence number is given in the source program as fixed, the actual data that is entered into the table at object program execution may still be composed of a variable number of occurrences of the table elements. Thus, not every table element must contain valid data.

To manipulate specific items and provide table searching, a technique called indexing is also available. Both subscripting and indexing are described below.

Table Definition

To define a one-dimensional table, an OCCURS clause is used as part of the data description of the table element, but the OCCURS clause must not appear in the description of group items which contain the table element. Example 1 shows a one-dimensional table defined by the item TABLE-ELEMENT.

Example 1:

```
02  TABLE-1.
    03  TABLE-ELEMENT; OCCURS 20 TIMES.
        04  DOG; ...
        04  FOX; ...
```

In the preceding example, the complete set of occurrences of TABLE-ELEMENT has been assigned the name TABLE-1. However, it is not necessary to assign a group name to the table unless it is desired to refer to the complete table as a group item.

Defining a one-dimensional table within each occurrence of an element of another one-dimensional table produces a two-dimensional table. To define a two-dimensional table, then, an OCCURS clause must appear in the data description of the element of the table, and in the description of only one group item which contains that element. Thus, in Example 2 below, DOG is an element of a two-dimensional table; it occurs five times within each element of the item BAKER which itself occurs 20 times. BAKER is an element of a one-dimensional table.

Example 2:

```
02  BAKER; OCCURS 20 TIMES; ...
    03  CHARLIE; ...
    03  DOG; OCCURS 5 TIMES; ...
```

In the general case, to define an n-dimensional table, the OCCURS clause should appear in the data description of the element of the table and in the descriptions of (n-1) group items that contain the element. In COBOL, tables of up to three dimensions are permitted; n cannot exceed three in the foregoing definition.

Initial Values of Tables

In the Working-Storage Section, initial values of elements within tables are specified in one of the following ways:

- The table may be described as a record by a set of contiguous data description entries, each of which specifies the VALUE of an element, or part of an element, of the table. In defining the record and its elements, any data description clause (USAGE, PICTURE, etc.) may be used to complete the definition where required. This form is required when the elements of the table need separate handling due to synchronization, usage, etc. The hierarchical structure of the table is then shown by using the REDEFINES entry and its associated subordinate entries. The subordinate entries (following the REDEFINES entry), which are repeated due to OCCURS clauses, must not contain VALUE clauses.

- When the elements of the table do not require separate handling, the VALUE of the entire table may be given in the entry defining the entire table. The lower level entries will show the hierarchical structure of the table; lower level entries must not contain VALUE clauses.

Reference to Table Items

Whenever the user refers to a table element or, if the table element is a group item, to the items within it, or to a condition-name associated with the element or with items contained within the element, the reference must indicate which occurrence of the element is intended. For access to a one-dimensional table, the occurrence number of the desired element provides complete information. For tables of more than one dimension, an occurrence number must be supplied for each dimension of the table. In Example 2 then, a reference to the 4th BAKER or the 4th CHARLIE would be complete, whereas a reference to the 4th DOG would not. To refer to DOG, which is an element of a two-dimensional table, the user must refer to, for example, the 4th DOG in the 5th BAKER.

Subscripting

One method by which occurrence numbers may be specified is to append one or more subscripts to the data-name. A subscript is an integer whose value specifies the occurrence number of an element within the group item that has the next lower level-number. The subscript can be represented either by a literal which is an integer or by a data-name which is defined elsewhere as a numeric elementary item with no character positions to the right of the assumed decimal point. In either case, the subscript, enclosed in parentheses, is written immediately following the name of the table element. A table element must include as many subscripts as there are dimensions in the table whose element is being referred to. That is, there must be a subscript for each OCCURS clause in the hierarchy containing the data-name, including the data-name itself.

Example 3:

```
02  BAKER; OCCURS 20 TIMES; ...
    03  CHARLIE; ...
    03  DOG; OCCURS 5 TIMES
        04  EASY; ...
        88  MAX; VALUE IS ...
        04  FOX; ...
            05  GEORGE; OCCURS 10 TIMES; ...
                06  HARRY; ...
                06  JIM; ...
```

In Example 3, references to BAKER and CHARLIE require only one subscript; references to DOG, EASY, MAX, and FOX require two; and references to GEORGE, HARRY, and JIM require three.

When more than one subscript is required, the subscripts are written in the order corresponding to the occurrence numbers in successively less inclusive dimensions of the data organization. If a multidimensional table is thought of as a series of nested tables and the most inclusive or outermost table in the nest is considered to be the major table with the innermost or least inclusive table being the minor table, then the subscripts are written from left to right in the order major, intermediate, and minor. Thus, in Example 3, a reference to HARRY (18 2 7) means the HARRY in the 7th GEORGE, in the 2nd DOG, in the 18th BAKER.

A reference to an item must not be subscripted if the item is not a table element or an item or condition-name within a table element.

The lowest permissible subscript value is one (1). The highest permissible subscript value in any particular case is the maximum number of occurrences of the item as specified in the OCCURS clause.

When a data-name is used as a subscript, it may be used to refer to items within many different tables. These tables need not have elements of the same size. The data-name may also appear as the only subscript with one item and as one of two or three subscripts with another item. It is also permissible to mix literal and data-name subscripts; for example, HARRY (12 NEWKEY 2).

Indexing

Another method of referring to items in a table is indexing. To use this technique, one or more index-names are assigned to an item whose data description contains an OCCURS clause. The INDEXED BY phrase, by which the index-name is identified and associated with its table, is an optional part of the OCCURS clause. There is no separate entry to describe the index-name since its definition is provided by the compiler and it is not considered data per se. At object program execution, the contents of the index-name will correspond to an occurrence number for that specific dimension of the table to which the index-name was assigned. The initial value of an index-name at object program execution is not determinable and the index-name must be initialized by the SET statement before use.

Example 4:

```
02  BAKER; OCCURS 20 TIMES; INDEXED BY IX-1; ...

    03  DOG; OCCURS 5 TIMES; INDEXED BY IX-2; ...

        05  GEORGE; OCCURS 10 TIMES; INDEXED BY IX-3; PIC XXXX; ...
```

In Example 4, references to BAKER require one subscript or the one index, IX-1; references to DOG require two subscripts or the two indexes IX-1, IX-2; and references to GEORGE require three subscripts or the three indexes IX-1, IX-2, IX-3.

An index-name may be used as an operand only by the SET, SEARCH, or PERFORM statements, or by the word IF in a relation condition. An index-name cannot be described as data within a COBOL program. Data items described by the USAGE IS INDEX clause permit storage of the values of index-names as data without conversion. Such data items are called index data items.

When a reference is made to a table element, or to an item within a table element, and the name of the item is followed by its related index-name(s) in parentheses, then each occurrence number required to complete the reference will be obtained from the respective index-name. The index-name thus acts as a subscript whose value is used in any table reference that specifies indexing.

When a reference requires more than one occurrence number for completeness, a data-name subscript must not be used to indicate one occurrence number and an index-name to indicate another. Therefore, if indexing is to be used, each OCCURS clause within the hierarchy (each dimension of the table) must contain an INDEXED BY phrase. The user may, however, mix literals and index-names within one reference, just as literals and data-name subscripts may be mixed.

When a statement that refers to an indexed table element is executed, the value of the index-name associated with the table element must not correspond to a value less than 1 nor to a value greater than the highest permissible subscript value for the table element.

The use of subscripting in a reference to a table element, or to an item within a table element, will not cause alteration of any index-names associated with that table.

Relative indexing is an additional option for making references to a table element or to an item within a table element. When the name of the table element is followed by an index in the form (index-name + integer-1), the occurrence number required to complete the reference will be the same as if integer-1 were added to the occurrence number to which the current setting of the index-name corresponds at object program execution. Similarly, when the form (index-name - integer-2) is used, the occurrence number obtained will be the same as if integer-2 were subtracted from the occurrence number to which the current setting of the index-name corresponds.

Relative indexing will not cause the object program to alter the value of the index-name.

A reference to an item must not be indexed by an index-name that is not associated (using the INDEXED BY phrase) with the table of which this item is an element.

Data that has been arranged in the form of a table is very often searched. In COBOL, the SEARCH statement provides facilities, through its two options, for producing serial and nonserial (i.e., binary) searches. In the SEARCH statement, the user may vary an associated index-name or an associated data-name. This statement also provides facilities for executing imperative-statements when certain conditions are true and an AT END phrase is included.

For additional information concerning table handling, refer to Section XIII of the COBOL User's Guide.

MASS STORAGE

The operational characteristics and the processing requirements of mass storage devices differ significantly from those of magnetic tape, punched paper tape, and punched cards. Tape and card files are normally organized in a sequential manner; the Data and Procedure Divisions of COBOL, prior to the inclusion of the mass storage facility, reflected these characteristics.

Mass storage media can be used to store sequentially organized files and this technique has been provided; more significantly, mass storage devices have been designed to provide nonsequential storage and access capabilities.

The mass storage feature of COBOL provides for the effective use of mass storage devices. Mass storage phrases are included in the Environment Division to describe the characteristics of mass storage files. The Procedure Division statement, SEEK, together with extensions to the OPEN, READ, WRITE, and CLOSE statements, provide facilities for efficiently processing mass storage files.

Access and Processing Techniques

The usual technique for applications using magnetic tape is sequential access to the data file and sequential processing of data records. This sequential-sequential technique is available for mass storage applications. Another technique for mass storage applications is called random access and sequential processing or the random-sequential technique. Either of these techniques may be specified by the user as the manner in which a particular mass storage file is to be processed.

A Mass Storage Control System provides the mechanism for control of these techniques.

Sequential Access with Sequential Processing

Although the sequential-access technique is similar in concept to the technique commonly used in processing magnetic tape files, a substantial difference exists between the physical environment of magnetic tape storage and the physical environment of mass storage.

In processing magnetic tape files, the execution of a READ statement implies the possibility of physical movement of the tape reel and proper positioning of the reel for subsequent READ statement executions. This positioning is done without regard for execution of WRITE statements that reproduce the updated input record onto a physically different output file. In processing mass storage files, READ statements may refer to the same physical file as the associated WRITE statements. That is, mass storage files are usually used for input and output at the same time. The usual file maintenance method is to read a record, process the record, and return it to its previous location by means of a WRITE statement. Thus, once a record is located and read from a mass storage file, the record location may be retained and, when the record is returned to the file by the execution of a WRITE statement, the execution time for the WRITE statement may be reduced.

An ACTUAL KEY phrase, which specifies the actual hardware location of a specific mass storage record, is not required for the sequential-sequential technique. However, if the ACTUAL KEY phrase is specified, varying the contents of the data item specified in the ACTUAL KEY phrase (actual key) will not result in any variation in processing order. In the sequential-sequential mode, the actual key is updated automatically by the Mass Storage Control System to reflect the location of the mass storage record currently being processed. Between the execution of the READ and WRITE statements for a particular file, the contents of the actual key are static.

The execution of a READ statement followed logically by the execution of a WRITE statement for the same input-output file results in an automatic updating of the actual key immediately after the execution of the WRITE statement. Similarly, the execution of a WRITE statement followed logically by the execution of another WRITE statement for the same file results in an automatic updating of the actual key after the execution of each WRITE statement. However, the execution of a READ statement followed logically by the execution of another READ statement from the same file, without the intervening execution of a WRITE statement, results in the automatic updating of the actual key only immediately prior to the execution of the second READ statement. Following the execution of a WRITE statement, the contents of the actual key reflect the actual location of the next mass storage record capable of being processed. In terms of COBOL logic, this is the location of the current mass storage record. Since the automatic updating of the contents of the actual key is the function of the Mass Storage Control System and since the ACTUAL KEY phrase is never referred to or required by the Mass Storage Control System, any changes the user makes to the actual key do not affect the processing of the mass storage file.

The imperative-statement in the AT END phrase associated with the next READ statement in order of execution is executed when the logical end of the mass storage file is detected. For WRITE statements, the detection of the logical end of a mass storage file before the execution of the CLOSE statement causes the contents of the actual key to reflect a location outside the environmental limits of the file. Since this value represents an erroneous location in the file, the INVALID KEY phrase associated with a particular WRITE statement is executed when that WRITE statement is executed.

Random Access with Sequential Processing

In the sequential-sequential technique, the data records in a mass storage file are read, processed, and written in an order based on the source program. The random-sequential technique differs only in that references are made to records in the file in a random manner. The sequential processing of randomly accessed records has all of the processing characteristics and file characteristics of the sequential-sequential method.

To permit direct access to any data record in a file, the user must specify a key to the precise identification of the particular record desired. **This identification must be specified as a single-precision binary integer (USAGE COMP-1).** Since, in this technique, the control of an actual key is the responsibility of the user, there are no implicit updating functions for an actual key.

In some computer systems, the introduction of the random-access approach to a mass storage file requires the definition of an input-output statement (SEEK) to operate in conjunction with the READ and WRITE statements. Since locating records is always necessary in the random-sequential technique, the function of the SEEK statement is performed implicitly by a READ or WRITE statement when the SEEK statement is not specified. The contents of the actual key are used by the Mass Storage Control System as the desired record's location identifier at the time the implicit SEEK statement is executed.

The SEEK statement, then, locates a record for subsequent reading or writing. On the computer it is not possible to separate physical seek times from the operations of reading or writing. Therefore, the SEEK statement is treated as a comment and no overlapping of other procedures is possible by its use prior to a READ or WRITE.

If the user has specified random access for a mass storage file, there is no logical end to the file. Thus, the AT END phrase of the READ statement is meaningless and the INVALID KEY phrase must be specified for both the READ and WRITE statements. If, during execution of either a READ or a WRITE statement, the contents of the actual key reflect an actual location outside the environmental limits for a file (as defined on system file control cards), the imperative-statement in the INVALID KEY phrase is executed.

For additional information concerning file processing, refer to Section V of the COBOL User's Guide.

SEGMENTATION

COBOL segmentation allows the user to communicate with the compiler to specify object program overlay requirements. Segmentation is concerned only with the segmentation of procedures. Therefore, only the Environment and Procedure Divisions are considered when determining segmentation requirements for an object program.

Terminology

With the advent of a standard concept of segmentation as an extension to COBOL, it has become necessary to revise terminology. Previously, the terms 'segmentation' and 'segments' were used to denote capabilities associated with the operating system. Where these words are used now, they will apply only to the standard COBOL concept of segmentation as discussed in this manual. The term modularization will replace the previous usage of the term segmentation and the term module will replace the term segment.

Modularization, then, is the facility of combining separately compiled programs (modules) as building blocks to the solution of a problem using a non-COBOL CALL statement to effect serial transfer of control. This concept is a Series 60/6000 feature and thus must be considered nonstandard.

Program Segments

The segmentation feature permits the user to subdivide the Procedure Division of a COBOL object program. All source paragraphs which contain the same priority-number in their section headers are considered to be one segment at object program execution. Since priority-numbers can range from 00 through 99, it is possible to subdivide any object program into a maximum of 100 segments.

The Procedure Division for a source program is usually written as a consecutive group of sections, each of which is composed of a series of closely related operations that are designed to collectively perform a particular function. When segmentation is used, the entire Procedure Division must be in sections. Each section must also be classified (using a priority-number) as belonging either to the fixed portion or to one of the independent segments of the object program. Segmentation in no way affects the need for qualification of procedure-names to ensure uniqueness.

Fixed Portion

The fixed portion is defined as that part of the object program which is logically treated as if it were always in memory. This portion of the program is composed of two types of segments: fixed permanent segments and fixed overlayable segments.

A fixed permanent segment is a segment in the fixed portion which cannot be overlayed by any other part of the program. A fixed overlayable segment is a segment in the fixed portion which, although logically treated as if it were always in memory, can be overlayed by another segment to optimize memory utilization. A fixed overlayable segment, if called for by the program, is always made available in its last used state.

Variation of the number of fixed permanent segments in the fixed portion can be accomplished by using a special facility called the SEGMENT-LIMIT phrase in the OBJECT-COMPUTER paragraph of the Environment Division. Unless the SEGMENT-LIMIT phrase is used, all segments numbered 00 through 49 are fixed permanent segments. However, if the user requires fixed overlayable segments, they are numbered from a user-specified value (01 through 49). The user indicates the lowest numbered segment which is to be fixed overlayable in the SEGMENT-LIMIT phrase. Therefore, the more fixed overlayable segments there are, the fewer fixed permanent segments there can be. Segment 00 is always fixed.

The logical relationship between all segments numbered 00 through 49 is always the same, regardless of SEGMENT-LIMIT. For example, an altered GO TO statement which appears in a segment numbered 27 will remain altered, whether the segment is fixed permanent or fixed overlayable, until the execution of another ALTER statement; intervening overlays of the segment will not result in initialization of the segment. Therefore, COBOL paragraphs numbered 00 through 49 can be written as if all such paragraphs were always fixed in memory, and subsequent changes in SEGMENT-LIMIT will have no effect on program logic.

Independent Segments

An independent segment is defined as part of the object program which can overlay, and can be overlayed by, either a fixed overlayable segment or another independent segment. An independent segment is effectively in its initial state each time the segment is made available to the program. Independent segments are numbered 50 through 99.

For additional information concerning segmentation, refer to Section XV of the COBOL User's Guide.

MODULARIZATION

Modules are programs that are compiled and tested independently and subsequently loaded together and executed as a total program. Thus, a user may decide to break up a large complex program into several parts or modules, write each one as a separate source program, and compile and test each module independently, thereby overlapping programming and checkout time. Another use of modules is to facilitate writing common subroutines (installation oriented) in source language to be compiled as independent modules.

The objectives of COBOL program modularization are:

1. To permit practical separation of a data processing program into distinct functional components (modules).
2. To permit the modules to be developed as separate COBOL source programs that are compiled separately and may be debugged separately.
3. To permit programs to be linked by the object program loader.
4. To permit the functional modules to overlay other modules when called into memory in order to execute large programs within a limited amount of memory.

The compiler provides segmentation of COBOL procedural sections as specified in American National Standard COBOL specifications. However, through modularization, it is possible to organize a group of COBOL modules functionally to operate in the same manner as segmentation of American National Standard COBOL-1968 procedural statements, with the exception that there is no way to call a segment and have it made available in its last used state. Refer to Section XV of the COBOL User's Guide for additional information.

USING A COBOL LIBRARY

A COBOL library containing source program text that is available at compilation may be created. By creating a library file, the user can avoid lengthy repetitions of data descriptions and/or procedures in programs using common data descriptions and/or procedures. The effect of the compilation of library text is the same as if the text were actually written as part of the source program. The library may contain text for the Environment Division, the Data Division, and the Procedure Division. **The compiler is directed to the library using the LIBCPY option or the COPY option on the $ COBOL card.** The text is made available by using the COPY clause in the Environment and Data Divisions and by using the COPY statement in the Procedure Division.

For a description of the COPY function, refer to Section VIII, the COBOL Library and to Section XIV of the COBOL User's Guide.

TRANSACTION PROCESSING

Transaction Processing System

The Transaction Processing System (TPS) gives a remote terminal user the ability to process business transactions on a large-scale computer. The transactions involved may represent any of the events in the minute-by-minute operation of a widespread, diversified business.

A subroutine in the General Comprehensive Operating Supervisor (GCOS) called the Transaction Processing Executive (TPE) calls Transaction Processing Applications Programs (TPAPs) whenever they are needed to process transactions from a remote terminal. This capability of dynamically selecting application programs to fit the incoming transaction allows the TPS to process an almost infinite variety of transactions.

Transaction Processing Applications Programs

Transaction Processing Applications Programs (TPAPs) are user-supplied programs designed to process the various types of transactions to be processed by the TPS. Thus, they are tailored to the precise needs of the user installation.

The TPAPs are stored as independent programs on the GCOS file system and are called into execution as needed to process transactions submitted to the TPS.

In addition to accepting transactions from a remote terminal, the TPAP may request direct communication with the user at the terminal and execute the transaction in a direct-access (DAC) mode. A wraparound provision in the TPS permits the output of one Transaction Processing Applications Program to be used as input to another Transaction Processing Applications Program for further processing.

Refer to the Transaction Processing System User's Guide for operational details of the TPS, and also to Section VII of the COBOL User's Guide.

SECTION III

LANGUAGE CONCEPTS

LANGUAGE RELATIONSHIPS

The English language is a natural language; that is, it is constantly changing and its rules describe current usage. COBOL is a mechanical language. Its rules have been predefined and are rigid, which means that the rules of COBOL must be changed before the use of COBOL can change.

As a mechanical language, COBOL is described as a higher level language. This means that COBOL is problem oriented, which has to do with the way data structures are defined. In lower level mechanical languages, the user must know the addressing structure of the computer as well as the way the computer structures data. In COBOL, knowledge of the computer's addressing structure is not required; data structures are defined according to the rules of COBOL rather than to the computer's rules. Because of this, COBOL is said to be a machine independent language.

Machine independence means that COBOL can be translated into many machine languages. A machine language is the terminology that a computer can recognize. The COBOL described in this manual is translated into the Series 60/6000 computer (machine) language.

Any language, natural or mechanical, is concerned with syntax and semantics. Syntax is the relationship between the symbols of the language. The semantics of a language are the relationships between the symbols of the language and their meanings. These relationships (and certain other considerations) make up the rules of the language. The remainder of this section describes the semantics of COBOL.

USER-CREATED SYMBOLS

The first part of this section is concerned with user-created COBOL symbols. Only three types of COBOL symbols can be created; words, literals, and PICTURE character-strings. All symbols created by the user are called character-strings. A character-string is defined as a contiguous set of characters taken from the COBOL character set that forms a word, a literal, or a PICTURE character-string. A character-string can be as short as one character in length or may have as many as 132 characters. A character is an element of the COBOL alphabet, which is known as the standard character set. The elements (characters) of the COBOL alphabet (character set) are:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9 + - * / = $, . ; " ( ) > < and

the blank or space.

There are 51 characters in the character set and only these characters can be used in character-strings.

Words

A COBOL word is a character-string of from one to 30 characters taken from the following subset of the COBOL character set:

A through Z

0 through 9

the special character '-' (hyphen)

When a COBOL word is formed, the special character '-' cannot be used as the first or last character of the word. A single COBOL word is often formed from two or more English language words. For example, COST-ANALYSIS is a single 13-character COBOL word and 6-AT-7-VAC-TUBE is a 15-character COBOL word. COBOL words are created by the user to name or identify something; COBOL names that can be created are data-names, condition-names, procedure-names, and mnemonic-names.

DATA-NAMES

The user creates a data-name to identify by name each data item described in the Data Division of the program or to identify by name the area that contains the data referred to in the Procedure Division. A data item can be an elementary item, a named group of elementary items within a record, or a record. An identifier is composed of a data-name, followed as required by a combination of qualifiers, subscripts, and/or indexes to make the identifier reference a unique data item in the program.

CONDITION-NAMES

A condition-name is a word that names one or more of the values a data item can have when the object program is being executed. A data item that can have more than one value while the object program is being executed is called a variable. A conditional variable is a data item whose value, sets of values, or range of values have names. Condition-names are used in the Procedure Division in conditional statements; i.e., in statements beginning with the word IF. Condition-names can be defined in the Data Division in level 88 entries or under SPECIAL-NAMES in the Environment Division. Each condition-name created by the user must be unique or be made unique through qualification.

PROCEDURE-NAMES

Procedure-names are words used to name paragraphs or sections in the Procedure Division of the program. Procedure-names enable the user to make references from one paragraph or section to another. This facility allows the user to write a procedure once and then refer to it as often as necessary. When a procedure-name is composed of digits such as 00345, a reference to that procedure-name must contain the leading zeros. That is, 345 is not the same as 00345.

MNEMONIC-NAMES

A mnemonic-name is a user-created COBOL word that is assigned to special names. Mnemonic-names are defined in the SPECIAL-NAMES paragraph of the Environment Division. The special names are names assigned by the computer system to certain input-output functions. Mnemonic-names do not have data descriptions in the Data Division.

Literals

The COBOL symbol type known as a literal is a user-created data item. That is, a literal is not a reference to data such as a data-name or identifier, but is the actual data to be operated on. A literal is a constant since its value never changes. The value of a literal is the characters that compose the literal. Two types of literals may be created, nonnumeric literals and numeric literals.

NONNUMERIC LITERALS

A nonnumeric literal can be either class alphabetic or class alphanumeric and can be used only as a display item. This means that nonnumeric literals cannot be used in computations; a nonnumeric literal cannot be added to some data item. A nonnumeric literal is defined as a string of characters in the Series 60/6000 character set, excluding the quotation mark character, bounded by quotation marks. The user can create a nonnumeric literal that has only one character or has as many as 132 characters, not counting the quotation marks that delimit it.

NUMERIC LITERALS

A numeric literal must be class numeric but, unlike nonnumeric literals, a numeric literal can be used as a computational item as well as a display item. To create a numeric literal, only the digits (0 through 9), the + sign, the - sign, and the decimal point may be used. No other characters are allowed in the construction of a numeric literal. When either the + or the - sign is used (only one of these can be used in each numeric literal), it must be the leftmost character of the literal. When neither sign is used, the literal is assumed to be positive. When the decimal point is used, it can appear in any character position of the literal except in the rightmost character position. If the decimal point is not used, the literal is an integer. Numeric literals may contain up to 18 digits and must contain at least one digit. The signs (+ and -) and the decimal point are not counted when applying this rule.

PICTURE Character-Strings

A PICTURE character-string is a special type of user-created COBOL symbol. The PICTURE character-strings that can be created are described in detail in the description of the PICTURE clause in the Data Division.

OTHER COBOL SYMBOLS

This paragraph is concerned with COBOL symbols that are used by but not created by the programmer. These symbols are Figurative Constants, Names of Special Registers, Editing Symbols, Punctuation Symbols, Relation Symbols, Arithmetic Operation Symbols, and Reserved Words.

Figurative Constants

A constant is a data item whose value remains fixed. Certain constants have been assigned fixed data-names, and are called figurative constants. A figurative constant may be used any place in the source program that a literal can be used except that wherever a numeric literal is required, the only figurative constant that can be employed is ZERO (or ZEROS or ZEROES). Although figurative constants may be specified in place of nonnumeric literals, this does not mean that figurative constants are delimited by quotation marks; they are not. Figurative constants are COBOL-defined symbols and are recognized as such by the compiler. Therefore, if a figurative constant is delimited by quotation marks, the result is a nonnumeric literal whose value is the word itself and not the value that the word implies. The figurative constants and their meanings are:

ZERO ZEROS ZEROES:	Represents the value 0 or one or more of the character 0, depending on the context in which it appears.
SPACE SPACES:	Represents one or more blanks or spaces.
UPPER-BOUND UPPER-BOUNDS:	Represents one or more of the character Z. The Z character is used as a high delimiter in processing data.
LOWER-BOUND LOWER-BOUNDS:	Represents one or more of the character 0. The 0 character is used as a low delimiter in processing data.
HIGH-VALUE HIGH-VALUES:	Represents one or more of the character !. The ! character has the highest value in the computer's collating sequence.
LOW-VALUE LOW-VALUES:	Represents one or more of the character 0. The 0 character has the lowest value in the computer's collating sequence.
QUOTE QUOTES:	Represents one or more of the character ". Note that this figurative constant cannot be used to delimit a nonnumeric literal.
ALL literal:	Represents one or more of the string of characters composing the literal. The literal must be either a nonnumeric literal or any other figurative constant. When the word ALL is followed by a figurative constant, the word ALL is redundant and is included in the source program only for readability.

The singular and plural forms of the figurative constants are equivalent and, therefore, may be used interchangeably. A figurative constant represents a string of characters. The number of characters in the string is determined by the compiler as follows:

1. When a figurative constant is associated with another data item, such as when the figurative constant is moved to or compared with another data item, the string of characters that the figurative constant represents is repeated character by character until the number of characters in the resulting string is equal to the number of characters specified as the size of the associated data item.

2. When a figurative constant is not associated with another data item, such as when the figurative constant appears in a DISPLAY, EXAMINE, or STOP statement, the length of the string is one character.

3. The ALL literal figurative constant cannot be used with the DISPLAY, EXAMINE, or STOP statements.

Special Registers

Special registers are compiler-generated memory areas. They are used to store information that is produced when specific COBOL features are used.

TALLY REGISTER

The primary use of the TALLY register is to temporarily store the information produced by the EXAMINE statement when the TALLYING option associated with it is used. The TALLY register generated is just large enough to contain a decimal integer of five digits whose implicit usage is COMP-1 and that has an operational sign. The word TALLY may also be used as the name of an elementary data item whose value is a decimal integer and whose size is not greater than five digits.

LINE-COUNTER

The purpose of the LINE-COUNTER is to automatically control the vertical positioning of a report. PAGE/OVERFLOW HEADING and PAGE/OVERFLOW FOOTING report groups are automatically produced based on the PAGE LIMIT clause specified in each report description entry. These report groups are used by the compiler to control the placement of headings and footings on the pages of the report(s) being generated. Although the LINE-COUNTER is generated by the compiler, its maximum value may be controlled using the PAGE LIMIT clause.

PAGE-COUNTER

The purpose of the PAGE-COUNTER is to supply page numbers for the pages within a report group. The initial value of the PAGE-COUNTER is one, but this value can be modified by the user immediately after an INITIATE statement has been executed.

Editing Symbols

The editing rules are described in Section VI under the PICTURE clause. This paragraph lists those characters of the COBOL character set that are editing symbols when used in an editing context. The symbols and their meanings are:

Symbol	Meaning
B	Space
0	Zero
+	Plus
-	Minus
CR	Credit
DB	Debit
Z	Zero Suppress
*	Check Protect
$	Currency Sign
,	Comma
.	Period (Decimal Point)

Punctuation Symbols

The punctuation symbols are used to delimit character-strings to make source coding more readable and, in certain instances, to conform to the requirements of the language. For example, the compiler normally expects to encounter a period followed by a space after a section-name. If the period or the space or both do not appear immediately after a section-name, an error condition results. For the most part, however, punctuation symbols are optional in the formats. When punctuation symbols are used, however, the following rules must be observed:

1. When used to delimit a character-string, the period, comma, or semicolon must be followed immediately by a space.

2. Whenever a space is used as a punctuation symbol, as many as desired may be used.

3. The left parenthesis may be immediately followed by a space and the right parenthesis may be immediately preceded by a space.

The punctuation symbols and their meanings are:

Symbol	Meaning
,	Comma
;	Semicolon
.	Period
(	Left Parenthesis
)	Right Parenthesis
	Space
"	Quotation Mark

Relation Symbols

Special symbols may be used in conditional statements in the Procedure Division to express a relationship (in value) between two data items. The symbols used for this purpose and their meanings are:

Symbol	Meaning
>	Greater Than
<	Less Than
=	Equal To

Arithmetic Operation Symbols

The following symbols may be used to perform arithmetic operations in the Procedure Division:

Symbol	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Reserved Words

Appendix A of this manual contains a list of words reserved for the exclusive use of COBOL. That is, each of these words is a COBOL symbol with a predefined meaning and must not appear as a user-defined word. Since the user can define data-names, condition-names, procedure-names, mnemonic-names, literals, and PICTURE character-strings, none of the reserved words can be used for any of these representations.

CONCEPT OF COMPUTER-INDEPENDENT DATA DESCRIPTION

The Data Division is that area of the source program in which the user describes the data that the object program is to produce or process. To make data as computer independent as possible, the attributes (characteristics or properties) of the data are described in relation to a standard data format rather than in relation to an equipment-oriented format. The data contained in a file is described according to the physical aspects of the file and the conceptual (or logical) characteristics of the data in the file.

Physical Aspects of a File

The physical attributes of a file describe the data as it appears on the input or output device. Some of the physical attributes of a file are:

1. The recording mode.

2. The grouping (or blocking) of logical records within the physical limitations of the device.

3. The means by which the file is identified.

These file attributes are described or defined in the Data Division of the source program.

Conceptual Characteristics of a File

The logical attributes of a file are an explicit definition of each logical entity in the file and are described or defined in the Data Division. In COBOL, a logical record is a group of related information that is uniquely identifiable and treated as a unit. A physical record, on the other hand, is a physical unit of information whose size and recording mode is convenient for a particular computer for the storage of data on an input or output device. The size of a physical record is hardware dependent and need bear no direct relationship to the size of the file of information on a device. For example, a single logical record may be contained in a single physical record; several logical records may be contained in a single physical record; or a single logical record may require several physical records to contain it. In COBOL, the input and output statements refer to one logical record. The idea of a logical record is not confined to data contained in files. Data in the Working-Storage Section can also be grouped into logical records.

Record Concepts

Each unique logical record is defined by a record description entry in the Data Division. Record description entries consist of data description entries that describe the attributes of a particular logical record. Each data description entry in a record description entry consists of a level-number which is followed by a data-name and by a series of independent clauses as required for the entry.

Concept of Levels

The structure of a logical record is based on levels so that subdivisions of the record can be named or identified for data reference. After a subdivision has been named, it can be further subdivided. The most basic subdivision of a record (that which is not further subdivided) is an elementary item. To refer to a set of elementary items, the items are combined into named groups. A group consists of a named sequence of one or more elementary items. Groups may also be combined into a sequence of two or more groups. An elementary item can, therefore, belong to more than one group. Note that if a logical record is not subdivided, it is an elementary item.

COBOL employs a system of level-numbers to show the hierarchical structure of logical records. Since a logical record is the most inclusive data item, level-numbers for records start at 01. Less inclusive data items (i.e., group items and elementary items) are assigned higher (but not necessarily successive) level-numbers not exceeding 49 in value. A group includes all group and elementary items following it until a level-number less than or equal to the level-number of that group is encountered. The level-number of an item (either an elementary item or a group item) which immediately follows the last elementary item of a preceding group must be the same as one of the groups to which the preceding elementary item belongs. In other words, level-numbers may not be assigned after a group structure has already been established by specific level-numbers.

The special level-numbers 66, 77, and 88 are exceptions to the rule that level-numbers cannot exceed 49 in value. These special level-numbers are not associated with describing the hierarchical structure of a logical record but, instead, are used to specify:

1. Elementary items or groups introduced by a RENAMES clause.
2. Noncontiguous working-storage data items and constants.
3. Condition-names.

Entries that describe items through a RENAMES clause (to regroup data items) use the level-number 66. Entries specifying noncontiguous data items (which are not subdivided and are not subdivisions of other items) use the level-number 77. Entries specifying condition-names (to be associated with particular values of a conditional variable) use the level-number 88.

Concept of Classes of Data

The five categories of data items (refer to the PICTURE clause) are grouped into three classes; alphabetic, numeric, and alphanumeric. For alphabetic and numeric, the classes and categories are synonymous. The alphanumeric class includes the categories of alphanumeric edited, numeric edited, and alphanumeric (without editing). Every elementary item belongs to one of the three classes and also to one of the categories. At object program execution, the class of a group item is treated as alphanumeric regardless of the class of elementary items subordinate to that group item. The following chart displays the relationship of the class and categories of data items.

LEVEL OF ITEM	CLASS	CATEGORY
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric Edited Alphanumeric Edited Alphanumeric
Nonelementary (Group)	Alphanumeric	Alphabetic Numeric Numeric Edited Alphanumeric Edited Alphanumeric

The size of an elementary data item or a group item is the number of characters in standard data format of the item. Synchronization and usage may cause a difference between this size and the character positions required for the internal representation of the data.

Algebraic Signs

Algebraic signs are used to indicate whether the value of a data item is positive or negative; two kinds of signs are employed:

1. An operational sign is used to show the value of an item in an operation.

2. An editing sign is used to identify the value of an item on an external, edited report.

The operational sign must be included in the last digit of a numeric item unless the data description entry specifies USAGE COMPUTATIONAL-4. In this case, the operational sign must be a separate four-bit digit immediately following the last digit of the numeric data item. If signs other than operational signs are used on input data, special handling will be required in the Procedure Division statements. Editing signs are not operational signs; they are inserted into a data item using the sign control symbols of the PICTURE clause.

Standard Alignment Rules

The standard rules for positioning data within an elementary item depend on the category of the receiving item. These rules are:

1. If the receiving data item is described as numeric:

   a. The data is aligned by decimal point and is moved to the receiving character positions with zero-fill or truncation on either end as required.

   b. When a decimal point is not explicitly specified, the data item is treated as if it had an assumed decimal point immediately following its rightmost character with zero-fill or truncation to the left, as required.

2. If the receiving data item is a numeric edited data item, the data moved to the edited data item is aligned by decimal point with zero-fill or truncation at either end as required within the receiving character positions of the data item, except where editing requirements cause replacement of the leading zeros.

3. If the receiving data item is alphanumeric (other than a numeric edited data item), alphanumeric edited, or alphabetic, the sending data is moved to the receiving character positions and aligned at the leftmost character position in the data item with space-fill or truncation to the right, as required.

If the JUSTIFIED clause is specified for the receiving item, these standard rules are modified as described in the JUSTIFIED clause.

Qualification

Every user-specified name that defines an element in a COBOL source program must be unique, either because no other name has the identical spelling and hyphenation, or because the name exists within a hierarchy of names such that references to the name can be made unique by mentioning one or more of the higher levels of the hierarchy. The higher levels are called qualifiers and the process that specifies uniqueness is called qualification. Enough qualification must be mentioned to make the name unique; however, it may not be necessary to mention all levels of the hierarchy. Within the Data Division, all data-names used for qualification must be associated with a level indicator or a level-number. Therefore, two identical data-names must not appear as entries subordinate to a group item unless they are capable of being made unique through qualification. In the Procedure Division, two identical paragraph-names must not appear in the same section.

In the hierarchy of qualification, names associated with a level indicator are the most significant; then those names associated with level-number 01; then names associated with level-number 02, ..., 49. A section-name is the highest (and the only) qualifier available for a paragraph-name. Thus, the most significant name in the hierarchy must be unique and cannot be qualified. Subscripted or indexed data-names and conditional variables, as well as procedure-names and data-names, may be made unique by qualification. The name of a conditional variable can be used as a qualifier for any of its condition-names. Regardless of the available qualification, no name can be both a data-name and a procedure-name.

Qualification is performed by following a data-name or a paragraph-name by one or more phrases composed of a qualifier preceded by IN or OF. IN and OF are logically equivalent.

The general formats for qualification are:

Format 1:

```
{data-name-1     }  [ {OF}             ]
{condition-name  }  [ {IN}  data-name-2 ] ...
```

Format 2:

```
paragraph-name  [ {OF}  section-name ]
                  {IN}
```

The rules for qualification are as follows:

1. Each qualifier must be of a successively higher level and within the same hierarchy as the name it qualifies.

2. The same name must not appear at two levels in a hierarchy.

3. If a data-name or a condition-name is assigned to more than one data item in a source program, the data-name or condition-name must be qualified each time it is referred to in the Environment, Data, and Procedure Divisions (except REDEFINES where, by definition, qualification is unnecessary).

4. A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to from within the same section.

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

6. A name can be qualified even though it does not need qualification; if there is more than one combination of qualifiers that ensures uniqueness, then any such set can be used.

Subscripting

Subscripts can be used only when reference is made to an individual element within a list or table of like elements that have not been assigned individual data-names. (Refer to the OCCURS clause in Section VI and Table Handling in Section II.)

The subscript can be represented either by a numeric literal that is an integer or by a data-name. The data-name must be a numeric elementary item that represents an integer. When the subscript is represented by a data-name, the data-name may be qualified but not subscripted.

The subscript may contain a plus sign. The lowest possible subscript value is one (1); this value points to the first element of the table. The next sequential elements of the table are pointed to by subscripts whose values are 2, 3, ... The highest permissible subscript value in any particular case is the maximum number of occurrences of the item as specified in the OCCURS clause.

The subscript, or set of subscripts, that identifies the table element is enclosed in parentheses immediately following the terminal space of the table element data-name. The table element data-name appended with a subscript is called a subscripted data-name. Although not required, a comma may separate subscripts in a series.

The format is:

data-name (subscript [, subscript] ...)

Some examples of writing subscripts are:

```
MOVE RATE (REGION, STATE, CITY) TO LISTINGS.
IF HEIGHT (10) IS GREATER THAN . . .
MULTIPLY PRICE (STOCK-NO) BY INVENTORY (STOCK-NO).
EXAMINE TRACT (REGION) REPLACING . . .
```

Indexing

References can be made to individual elements within a table of like elements by specifying indexing for that reference. An index is assigned to that level of the table by using the INDEXED BY phrase in the definition of a table. A name given in the INDEXED BY phrase is known as an index-name and is used to refer to the assigned index. An index-name must be initialized by a SET statement before it is used as a table reference (refer to the SET statement, Section VII).

Direct indexing of a table element is specified by using an index-name in the form of a subscript. Relative indexing is specified when the index-name is followed by the operator + or -, followed by an unsigned integral numeric literal all enclosed in the set of parentheses that begins immediately after the terminal space of the data-name.

The general format for indexing is:

```
data-name  ( index-name [ {+} integer ]
                        {-}
   [, index-name [ {+} integer ] ]  ... )
                  {-}
```

Identifier

An identifier is a term used to reflect that a data-name (if not unique in a program) must be followed by a syntactically correct combination of qualifiers, subscripts, or indexes necessary to ensure uniqueness.

The general formats for identifiers are:

Format 1:

```
data-name-1 [ {OF} data-name-2 ] ... [ (subscript-1
              {IN}
   [, subscript-2 [ , subscript-3]]  ) ]
```

Format 2:

```
data-name-1 [ {OF} data-name-2 ] ...
              {IN}
   [ (index-name-1 [ {+} integer ]
                   {-}
   [ , index-name-2 [ {+} integer ]
                      {-}
   [ , index-name-3 [ {+} integer ]]]  ) ]
                    {-}
```

Restrictions on Qualification, Subscripting, and Indexing

The restrictions on qualification, subscripting, and indexing follow:

1. The commas shown in the general formats are not required.

2. A data-name must not itself be subscripted nor indexed when that data-name is being used as an index, subscript, or qualifier.

3. Indexing is not permitted where subscripting is not permitted.

4. An index may be modified only by the SET, SEARCH, and PERFORM statements. Data items described by the USAGE IS INDEX clause permit storage of the values associated with index-names as data. Such data items are called index data items.

5. If indexing is to be used, each OCCURS clause within the hierarchy (each dimension of the table) must contain an INDEXED BY phrase.

6. When a reference requires more than one occurrence number for completeness, a data-name subscript must not be used to indicate one occurrence number when an index-name is used for another.

7. Literals and index-names may be mixed within one reference just as literals and data-name subscripts may be mixed.

REFERENCE FORMAT

The reference format, which provides a standard method for describing COBOL source programs, is expressed in terms of character positions in a line on the input-output device. The COBOL compiler accepts source programs written in reference format and produces an output listing in reference format.

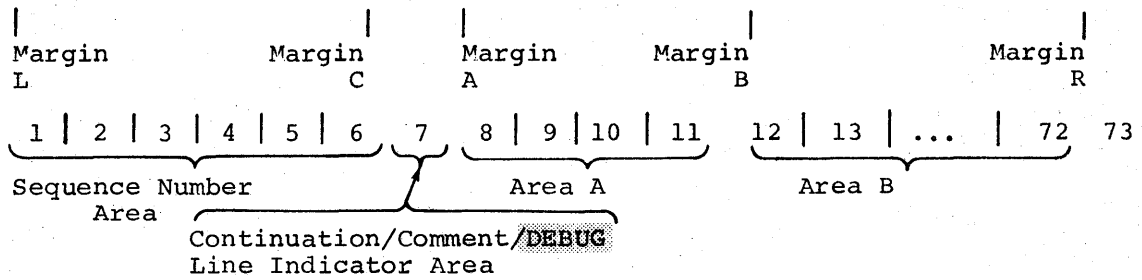
The rules for spacing given in the discussion of the reference format take precedence over all other rules for spacing.

The number of cards in a COBOL source program may not exceed 32,767.

The divisions of a source program must be ordered as follows: the Identification Division, the Environment Division, the Data Division, and the Procedure Division. Each division must be written according to the rules for the reference format.

Reference Format Representation

The reference format for a line is represented as follows:



Margin L designates the leftmost character position of a line, position 1.

Margin C designates the seventh character position relative to L.

Margin A designates the eighth character position relative to L.

Margin B designates the twelfth character position relative to L.

Margin R designates the rightmost character position of a line and is character position 72 relative to Margin L.

The sequence number area occupies the six character positions beginning at Margin L, positions 1 through 6.

The continuation/comment/DEBUG line indicator area is the seventh character position of the line.

Area A occupies four character positions beginning at Margin A, positions 8 through 11.

Area B occupies sixty-one character positions beginning at Margin B, positions 12 through 72.

SEQUENCE NUMBERS

A sequence number, consisting of six digits in the sequence number area, may be used to label a source program line.

CONTINUATION OF LINES

Any sentence or entry that requires more than one line is continued by starting subsequent line(s) in Area B. These subsequent lines are called the continuation line(s). The line being continued is called the continued line. Any word or literal may be broken in such a way that part of it appears on a continuation line.

A hyphen in the continuation area of a line indicates that the first nonblank character in Area B of the current line is the successor of the last nonblank character of the preceding line with no intervening space. However, if the continued line contains a nonnumeric literal without a closing quotation mark, the first nonblank character in Area B of the continuation line must be a quotation mark, and the continuation starts with the character immediately after that quotation mark. All spaces at the end of the continued line are considered part of the literal. Area A of a continuation line must be blank.

If no hyphen is contained in the continuation area of a line, it is assumed that the last character in the preceding line is followed by a space.

BLANK LINES

A blank line is one that is blank from Margin C to Margin R, inclusive. A blank line can appear anywhere in the source program except:

a. Immediately preceding a continuation line.

b. Immediately following a section header in a segmented program.

c. Immediately following a COPY statement.

Division, Section, and Paragraph Formats

DIVISION HEADER

The division header must start in Area A. After the division header, no text may appear before the following section header or paragraph header or paragraph-name, except that the keyword DECLARATIVES followed by a period may be present after the Procedure Division header.

SECTION HEADER

The section header must start in Area A.

A section consists of paragraphs in the Environment, Data, and Procedure Divisions. In the Environment and Data Divisions, no text may appear before the following paragraph header or paragraph-name.

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a priority-number (optional), followed by a period and a space.

PARAGRAPH HEADER, PARAGRAPH-NAME, AND PARAGRAPH

A paragraph consists of a paragraph-name followed by zero, one, or more sentences, or a paragraph header followed by one or more entries. A paragraph header starts in Area A of any line following the first line of a division or a section.

The name of a paragraph starts in Area A of any line following the first line of a division or a section and ends with a period followed by a space.

The first sentence or entry in a paragraph begins either on the same line as the paragraph-name or in Area B of the next nonblank line that is not a comment line. Successive sentences or entries either begin in Area B of the same line as the preceding sentence or entry or in Area B of the next nonblank line that is not a comment line.

A sentence consists of one or more statements; an entry consists of one or more clauses; all sentences and entries must be followed by a period followed by a space.

When the sentences or entries of a paragraph require more than one line, they may be continued as described in the Continuation of Lines paragraph above.

Data Division Entries

Each Data Division entry begins with a level indicator or a level-number, followed by a space, followed by the name of a data item (except in the Report Section), followed by a sequence of independent clauses describing the data item. Each clause, except the last clause of an entry, may be terminated by a semicolon followed by a space. The last clause is always terminated by a period followed by a space. There are two types of Data Division entries; those which begin with a level indicator and those which begin with a level-number.

A level indicator is any of the following: file description (FD), sort-merge file description (SD), or report description (RD). See Section VI.

In Data Division entries that begin with a level indicator, the level indicator begins in Area A followed in Area B by its associated data-name and appropriate descriptive information.

Data Division entries that begin with level-numbers are called data description entries.

In data description entries that begin with a level-number 01 or 77, the level-number begins in Area A followed by a space and its associated record-name or item name and appropriate descriptive information.

A level-number may be one of the following set: 01 through 49, 66, 77, 88. Single digit level-numbers are written either as a digit or as a zero followed by a digit. At least one space must separate a level-number from the word following the level-number.

Successive data description entries may have the same format as the first or may be indented according to level-number. The entries in the output listing are indented only if the input is indented. Indentation does not affect the magnitude of a level-number.

When level-numbers are to be indented, each new level-number may begin any number of spaces to the right of Margin A. The extent of indentation to the right is determined only by the width of the physical medium.

Declaratives

The keyword DECLARATIVES and the keywords END DECLARATIVES that precede and follow, respectively, the declarative portion of the Procedure Division must appear on a line by themselves. Each must begin in Area A and be followed by a period and a space. After the keywords END DECLARATIVES, no text may appear before the following section header.

Comment Lines

A comment line is any line with an asterisk in the continuation area of the line. A comment line can appear as any line in a source program, excluding:

1. The last line.
2. The line immediately following a section header in a segmented program.
3. The line immediately following a COPY statement.
4. Any line in an ENTER GMAP procedure.

Any combination of the characters from the computer's character set may be included in Area A and Area B of a comment line. The asterisk and the characters in Area A and Area B will be produced on the output listing but serve as documentation only. Four special types of comment lines are provided:

1. An * in column 7 followed by the word EJECT in columns 8-12 causes page ejection after printing the comment.
2. A stroke (/) in column 7 causes page ejection prior to printing the comment line.
3. An * in column 7 followed by LSTOF in columns 8-12 causes suppression of the COBOL source program listing.
4. An * in column 7 followed by LSTON causes termination of the suppression initiated by *LSTOF.

Debugging statements are identified as such by a single digit (0-9) in column 7 of the reference format. When statements are identified as debugging statements and the PROCESS DEBUG STATEMENTS option of the SPECIAL-NAMES paragraph is not included in the source program, those statements with a digit in column 7 are treated as comment lines.

The continuation of comment lines using a hyphen in column 7 of the next record is not permitted, but successive comment lines are allowed.

Definition of a General Format

A general format is the specific arrangement of the elements of a clause or a statement (which consist of elements as defined below). In this manual, a format is shown adjacent to information which defines a clause or statement. If more than one specific arrangement is permitted, the general format is separated into numbered formats. Clauses must be written in the sequence given in the general formats. (If used, optional clauses must appear in the sequence shown.) In certain cases, stated explicitly in the rules associated with a given format, clauses may appear in sequences other than those shown. Applications, requirements, or restrictions are presented as rules:

- A syntax rule amplifies or restricts the usage of the elements within a general format.
- A general rule amplifies or restricts functions attributed to a general format or to its constituent elements.

Elements which make up a clause or a statement consist of uppercase words, lowercase words, level-numbers, brackets, braces, connectives, and special characters.

Words

All underlined uppercase words are called keywords and are required when the functions of which they are a part are used. Uppercase words which are not underlined are optional and may be written in the source program at the discretion of the user. Uppercase words, whether underlined or not, must be spelled correctly and must appear in the source program exactly as shown in the formats.

In a general format, lowercase words are generic terms used to represent COBOL words that must be supplied by the user. Lowercase words which occur in a general format are replaced by COBOL words in an actual program, except for the following list of words:

- Statements (Section VII)
- Imperative-statements (Section VII)
- Arithmetic-expressions (Section VII)
- Character-strings (Section VI)
- Comment-entries (Section IV)
- Conditions (Section VII)
- Literals (Section III)

The above exceptions represent combinations of COBOL words constructed in accordance with the definitions given in the sections noted.

When generic terms are repeated in a general format, a number or letter appended to the term serves to identify that term for subsequent explanation.

Periods

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

Level-Numbers

When specific level-numbers appear in data description entry formats, those specific level-numbers are required when such entries are used in a COBOL program.

Brackets and Braces

When a portion of a general format is enclosed in brackets [ ], that portion may be included or omitted at the user's choice. Braces { } enclosing a portion of a general format indicate that one of the options contained within the braces must be selected. In both cases, the possible choices are stacked vertically in the format. When brackets or braces enclose a portion of a format but only one possibility is shown, the function of the brackets or braces is to delimit that portion of the format to which a following ellipsis applies (see next paragraph). If an option within braces does not contain a key word, that option is a default option (implicitly selected unless explicitly overridden).

The Ellipsis (...)

The ellipsis may show the omission of a portion of a source program. The meaning becomes apparent in context.

In the general format, the ellipsis represents the position at which repetition may occur at the user's option. The portion of the format that may be repeated is determined as follows:

> In a clause or statement format in which the ellipsis (...) appears, scan from right to left to determine the] or } delimiter immediately to the left of the ...; continue scanning from right to left and determine the logically matching [or { delimiter; the ... applies to the words between the pair of delimiters.

Format Punctuation

The punctuation characters comma and semicolon are shown in some formats. However, a semicolon must not appear immediately preceding the first clause of an entry or a paragraph. These punctuation characters are used in each major COBOL division as explained below.

- Identification Division - Although not expressly shown in the formats in this division, the comma and semicolon may be used within the comment-entries. The paragraph itself must terminate with a period followed by a space.
- Environment Division - When either a comma or a semicolon is shown in the formats, it is optional and may be included or omitted. The entry itself must terminate with a period followed by a space.
- Data Division - When either a comma or a semicolon is shown in the formats, it is optional and may be included or omitted. The entry itself must terminate with a period followed by a space.
- Procedure Division - When a comma is shown in the formats, the comma is optional and may be included or omitted. If desired, a semicolon may be used between statements.

Special Characters

When the characters '+', '-', '<', '>', and '=' appear in formats, they are required when such formats are used, even though they are not underlined.

Shading

In this manual, the shaded areas represent the implementation of a feature specified in the CODASYL COBOL Journal of Development but not specified in American National Standard COBOL-1968, the implementation of a feature defined in earlier versions of the COBOL language specifications but subsequently deleted by CODASYL, or the implementation of a feature that may be unique to the Series 60/6000 compiler. Some of these features are now contained in American National Standard COBOL-1974.

For additional information regarding obsolete language elements, refer to Section XVII of the COBOL User's Guide.

For information concerning the flagging of obsolete language elements and Series 60/6000 language extensions, refer to Appendix C of the COBOL User's Guide.

Deltas

A delta ▲ in the margin adjacent to the text indicates that the particular feature being described has not been implemented. The ▲ may appear adjacent to a feature not implemented or adjacent to a feature which is not available for use with the Software Release associated with this revision of the manual.

SECTION IV

IDENTIFICATION DIVISION

DESCRIPTION OF THE IDENTIFICATION DIVISION

Each COBOL source program must begin with the Identification Division, which is used to identify the source program and its resultant output listing. The user may also include the date on which the program is written, the date of source program compilation, and other desired information as shown in the paragraph structure of the general format below.

Organization of the Identification Division

Fixed paragraph-names specify the type of information contained in each paragraph. The name of the program may be given in the first paragraph, the PROGRAM-ID paragraph. The other paragraphs are optional and may be included at the discretion of the user, in any order.

Structure of the Identification Division

The structure of the Identification Division is presented below.

General Format:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. program-name.
[ AUTHOR. [comment-entry ] ...]
[ INSTALLATION. [comment-entry ] ...]
[ DATE-WRITTEN. [comment-entry ] ...]
[ DATE-COMPILED. [comment-entry ] ...]
[ SECURITY. [comment-entry ] ...]
[ REMARKS. [comment-entry ]...]
```

Syntax Rule:

1. The Identification Division must begin with the reserved words IDENTIFICATION DIVISION followed by a period and a space.

General Rule:

1. A comment-entry may be any combination of characters from the computer's character set, organized to conform to sentence and paragraph format.

PROGRAM-ID PARAGRAPH

The PROGRAM-ID paragraph is required and may be the first paragraph in the Identification Division.

General Format:

PROGRAM-ID. program-name.

Syntax Rules:

1. The PROGRAM-ID paragraph must begin with the paragraph-name PROGRAM-ID which must be followed by a period and a space. It must appear in every program.
2. The program-name supplied can be composed only of letters and digits. The program-name should not exceed six characters and must contain at least one letter. The first character of the program-name may not be a zero. The program-name cannot be any of the COBOL reserved words. It should also be different from any user-defined data-name. If the program-name exceeds six characters, it will be truncated. Any invalid characters will be replaced by a period.
3. The program-name must be followed by a period and a space.
4. The first four characters of program-name must not be LDIN.
5. If no program-name is specified, the compiler will supply the word NONAME as a substitute program-name.

General Rules:

1. The first three characters are used as the Transaction Processing Applications Program (TPAP) identifier (ID) when the program is to be used in the Transaction Processing System; therefore, they must be unique within that system (refer to the COBOL User's Guide).
2. When the program is to be loaded into the same overlay as other COBOL programs, the first four characters of each program-name must be unique within the overlay.
3. The program-name will appear on the program listing.

AUTHOR PARAGRAPH

The AUTHOR paragraph is used to supply the name of or otherwise identify the author of the program.

General Format:

AUTHOR. [comment-entry]...

Syntax Rules:

1. The AUTHOR paragraph must begin with the paragraph-name AUTHOR which must be followed by a period and a space.
2. The comment-entry can be one or more sentences, including any combination of characters from the computer's character set, organized to conform to sentence and paragraph format.

General Rules:

1. The paragraph is optional.
2. If the AUTHOR paragraph is included in the source program, the information supplied will appear on the program listing.

INSTALLATION PARAGRAPH

The INSTALLATION paragraph is used to supply the name of or otherwise identify the installation at which the source program was written.

General Format:

INSTALLATION. [comment-entry] ...

Syntax Rules:

1. The INSTALLATION paragraph must begin with the paragraph-name INSTALLATION which must be followed by a period and a space.
2. The comment-entry can be one or more sentences, including any combination of characters from the computer's character set, organized to conform to sentence and paragraph format.

General Rules:

1. The paragraph is optional.
2. If the INSTALLATION paragraph is included in the source program, the information supplied will appear on the program listing.

DATE-WRITTEN PARAGRAPH

The DATE-WRITTEN paragraph is used to supply the date on which the program was written.

General Format:

DATE-WRITTEN. [comment-entry] ...

Syntax Rules:

1. The DATE-WRITTEN paragraph must begin with the paragraph-name DATE-WRITTEN which must be followed by a period and a space.

2. The comment-entry can be one or more sentences, including any combination of characters from the computer's character set, organized to conform to sentence and paragraph format.

General Rules:

1. The paragraph is optional.

2. If the DATE-WRITTEN paragraph is included in the source program, the information supplied will appear on the program listing.

DATE-COMPILED PARAGRAPH

The DATE-COMPILED paragraph is used to supply the date on which the program was compiled.

General Format:

DATE-COMPILED. [comment-entry] ...

Syntax Rules:

1. The DATE-COMPILED paragraph must begin with the paragraph-name DATE-COMPILED which must be followed by a period and a space.

2. The comment-entry may be any combination of characters from the computer's character set, organized to conform to sentence and paragraph format.

General Rules:

1. The paragraph is optional.

2. If the DATE-COMPILED paragraph is included in the source program, it is replaced during compilation with a paragraph of the form:

   DATE-COMPILED. current-date.

SECURITY PARAGRAPH

The SECURITY paragraph is used to supply the level of security attached to the program by the installation or user.

General Format:

SECURITY. [comment-entry] ...

Syntax Rules:

1. The SECURITY paragraph must begin with the paragraph-name SECURITY which must be followed by a period and a space.
2. The comment-entry may be any combination of characters from the computer's character set, organized to conform to sentence and paragraph format.

General Rules:

1. The paragraph is optional.
2. If the SECURITY paragraph is included in the source program, the information supplied will appear on the program listing.

REMARKS PARAGRAPH

The REMARKS paragraph is used to supply any information about the program that is not contained in the other paragraph headings in the Identification Division.

General Format:

REMARKS. [comment-entry] ...

Syntax Rules:

1. The REMARKS paragraph must begin with the paragraph-name REMARKS which must be followed by a period and a space.
2. The information supplied can be one or more sentences, including any combination of characters from the computer's character set, organized to conform to sentence and paragraph format.
3. All lines of comment-entry are restricted to Area B of the reference format.

General Rules:

1. The paragraph is optional.
2. If the REMARKS paragraph is included in the source program, the information supplied will appear on the program listing.

SECTION V

ENVIRONMENT DIVISION

DESCRIPTION OF THE ENVIRONMENT DIVISION

The Environment Division must be included in every COBOL source program and is the second division of a COBOL program. This division provides a standard method of expressing the aspects of a data processing problem that depend upon the physical characteristics of any given computer. It is used to identify the compiling computer and the computer on which the object program is to be run. Data concerning input-output control, specific hardware characteristics, and control techniques can also be presented in this division.

Organization of the Environment Division

The Environment Division is divided into two sections, the Configuration Section and the Input-Output Section. The Configuration Section is required and the Input-Output Section is optional. The Configuration Section is subdivided into the following three paragraphs:

- SOURCE-COMPUTER paragraph, which identifies the computer on which the source program is to be compiled.
- OBJECT-COMPUTER paragraph, which identifies the computer on which the object program produced by the compiler is to be executed.
- SPECIAL-NAMES paragraph, which associates the names of hardware and operating system features used by the compiler with the mnemonic-names used in the source program.

The Input-Output Section is subdivided into the following two paragraphs:

- FILE-CONTROL paragraph, which names all files used in the program and associates them with external media.
- I-O-CONTROL paragraph, which defines special control techniques to be used in the object program.

Structure of the Environment Division

The general outline of the sections and paragraphs in the Environment Division and the order of presentation in the source program is given below.

General Format:

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

[SOURCE-COMPUTER. [source-computer-entry]]

[OBJECT-COMPUTER. [object-computer-entry]]

[SPECIAL-NAMES.   special-names-entry]

[INPUT-OUTPUT SECTION.

 FILE-CONTROL. { file-control-entry } ...

[I-O-CONTROL.   input-output-control-entry]]
```

Syntax Rules:

1. The Environment Division must be included, must follow the Identification Division, and must begin with the reserved words ENVIRONMENT DIVISION which must be followed by a period and a space.

2. The Configuration Section should be included, follow the Environment Division header, and begin with the section-name CONFIGURATION SECTION which must be followed by a period and a space.

CONFIGURATION SECTION IN THE ENVIRONMENT DIVISION

The Configuration Section provides program documentation for the hardware characteristics of the computer used for compilation and of the computer used to execute the object program. Provisions are included in this section for assigning definitions to all mnemonic-names to be used in the body of the program, for defining condition-names for the status of switches, and for defining specific compiler-directing phrases.

SOURCE-COMPUTER

SOURCE-COMPUTER Paragraph

The SOURCE-COMPUTER paragraph identifies the computer upon which the program is to be compiled. The paragraph can be written in either of the following two formats.

Format 1:

```
[ SOURCE-COMPUTER.   COPY   library-name
    [ REPLACING   word-1   BY { word-2 }
                              { literal-1 }
                              { identifier-1 }
      [ , word-3   BY { word-4 }                ] ... ] . ]
                      { literal-2 }
                      { identifier-2 }
```

Format 2:

```
[ SOURCE-COMPUTER.   [ 6000      ]
                     [ 6000-EIS  ]
  [WITH SUPERVISOR CONTROL]
  [ , MEMORY SIZE { integer-1 { WORDS      }                                    }
                  {           { CHARACTERS }                                   }
                  {           { MODULES    }                                   }
                  { ADDRESS literal-1 { THRU    } literal-2                    }
                  {                   { THROUGH }                            }
                  {     [ , literal-3 { THRU    } literal-4 ] ...              } ]
                  {                   { THROUGH }                            }
    [ , [integer-2] hardware-name-1 ] ... . ]
```

Syntax Rules:

1. The SOURCE-COMPUTER paragraph must begin with the paragraph-name SOURCE-COMPUTER which must be followed by a period and a space.

2. When Format 1 is used, the COPY library-name phrase is required. The library-name must be identical to the name associated with the desired text on the library.

3. In Format 1, a word is any COBOL word.

4. When Format 2 is used, the computer-name specified should be 6000 or 6000-EIS. Series 60 users should specify 6000-EIS.

5. The WITH SUPERVISOR CONTROL phrase is optional. When used, however, the word SUPERVISOR is required.

6. The MEMORY SIZE phrase is optional. When used, the word MEMORY is required. If the memory size is given as an integer, the integer specifies the number of WORDS, CHARACTERS, or MODULES required for compilation. Therefore, one of these three words is required when the memory size is given as an integer. If the memory size is given as an address, the word ADDRESS and either of the words THRU or THROUGH are required. The literals specify the required hardware locations needed for compilation.

7. The number and type of peripheral units used during compilation can be specified. The integer associated with a hardware-name specifies the number of units. The hardware-names may be any of the following:

MAGNETIC TAPE UNIT(S)	CARD PUNCH(ES)
DISC STORAGE UNIT(S)	CARD READER(S)
HIGH SPEED PRINTER(S)	TYPEWRITER(S)

More peripheral equipment than actually is needed may be specified.

General Rules:

1. For a description of the COPY function, see Section VIII, the COBOL Library.

2. This paragraph provides program documentation only and has no effect on compilation.

Special Considerations:

1. For a program that might be used in an installation with different computers, the SOURCE-COMPUTER paragraph can supply valuable information for modifying the program for the different environment.

OBJECT-COMPUTER Paragraph

The OBJECT-COMPUTER paragraph identifies the computer on which the program is to be executed. The paragraph can be written in either of the following two formats.

Format 1:

```
[ OBJECT-COMPUTER.   COPY   library-name
  [ REPLACING  word-1  BY  { word-2      }
                           { literal-1   }
                           { identifier-1 }
      [ , word-3  BY { word-4      } ] ... ] . ]
                     { literal-2   }
                     { identifier-2 }
```

Format 2:

```
[ OBJECT-COMPUTER. [ 6000      ]
                   [ 6000-EIS  ]

  [WITH SUPERVISOR CONTROL]

  [ , MEMORY SIZE { integer-1 { WORDS      }                                    }
                  {           { CHARACTERS }                                   }
                  {           { MODULES    }                                   }
                  { ADDRESS literal-1 { THRU    } literal-2                    }
                  {                   { THROUGH }                              }
                  {   [ , literal-3 { THRU    } literal-4 ] ...                }
                  {                 { THROUGH }                                } ]

  [ , [integer-2] hardware-name-1 ] ...

  [ , SEGMENT-LIMIT IS priority-number ] . ]
```

Syntax Rules:

1. The OBJECT-COMPUTER paragraph must begin with the paragraph-name OBJECT-COMPUTER which must be followed by a period and a space.

2. When Format 1 is used, the COPY library-name phrase is required. The library-name must be identical to the name associated with the desired text on the library.

3. In Format 1, a word is any COBOL word.

4. When Format 2 is used, the computer-name specified should be 6000 or 6000-EIS. Series 60 users should specify 6000-EIS.

5. The WITH SUPERVISOR CONTROL phrase is optional. When used, however, the word SUPERVISOR is required.

6. The MEMORY SIZE phrase is optional. When used, the word MEMORY is required. If the memory size is given as an integer, the integer specifies the number of WORDS, CHARACTERS, or MODULES the object program requires. Therefore, one of these three words is required when the memory size is given as an integer. If the memory size is given as an address, the word ADDRESS and either of the words THRU or THROUGH are required. The literals specify the required hardware locations needed for the object program.

7. The number and type of peripheral units used to execute the object program can be specified. The integer associated with a hardware-name specifies the number of units. The hardware-names may be any of the following:

MAGNETIC TAPE UNIT(S)	CARD PUNCH(ES)
DISC STORAGE UNIT(S)	CARD READER(S)
HIGH SPEED PRINTER(S)	TYPEWRITER(S)

More peripheral equipment than actually is required may be specified.

8. When the SEGMENT-LIMIT IS phrase is used, the priority-number must be an integer in the range 1-49. The words SEGMENT-LIMIT IS are required.

9. When the SEGMENT-LIMIT phrase is specified, only those segments having priority-numbers from 0 up to, but not including, the priority-number designated as the SEGMENT-LIMIT, are considered as permanent segments of the object program.

10. Those segments having priority-numbers from the SEGMENT-LIMIT through 49 are considered as fixed overlayable segments.

11. When the SEGMENT-LIMIT phrase is omitted, all segments having priority-numbers from 0 through 49 are considered as permanent segments of the object program.

General Rules:

1. For a description of the COPY function, see Section VIII, the COBOL Library.

2. If the EISF or NEISF options are specified on the $ COBOL card, these options will take precedence over the OBJECT-COMPUTER paragraph. A warning message will be given when EISF has been specified on the $ COBOL card and the following is encountered in the source program:

   OBJECT-COMPUTER. 6000.

   The object program will utilize EIS code.

   A warning message will be given when NEISF has been specified on the $ COBOL card and the following is encountered in the source program:

   OBJECT-COMPUTER. 6000-EIS.

   The object program will not utilize EIS code.

   If the OBJECT-COMPUTER paragraph is not present or a syntax error on computer-name (none present or illegal name) is detected, the compiler will determine whether or not EISF or NEISF was specified on the $ COBOL card before the default determination of processor type is made. If neither EISF nor NEISF were specified on the $ COBOL card, the object program will be prepared for the type of processor upon which the program is being compiled and a warning message will be issued.

Special Considerations:

1. The 6000-EIS phrase causes coding that utilizes the Extended Instruction Set (EIS)[1] to be produced unless overridden by the NEISF option on the $ COBOL card. An object program that includes EIS can only be executed on an EIS processor.

2. Programs in the same run unit should either all be compiled in the EIS mode or all be compiled in the NEIS mode; otherwise, unpredictable results (including abnormal terminations) may occur during object program execution.

3. For a program that might be used in an installation with different computers, the OBJECT-COMPUTER paragraph can supply valuable information for modifying the program for the different environment.

[1]Extended Instruction Set (EIS) refers to an extension to the original Series 6000 instruction set. EIS is the standard instruction repertoire for models 6025, 6040, 6060, and 6080 of the Series 6000 system and all models of the Series 60 system.

SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph is optional. It provides a method by which mnemonic-names used in the Data and Procedure Divisions, condition-names for the status of switches, and certain compiler-directing phrases can be specified. The paragraph can be written in either of the following two formats.

Format 1:

```
[ SPECIAL-NAMES. COPY   library-name

    [ REPLACING   word-1   BY { word-2      }
                              { literal-1   }
                              { identifier-1 }

        [ , word-3   BY { word-4      } ] ... ] . ]
                        { literal-2   }
                        { identifier-3 }
```

Format 2:

```
[ SPECIAL-NAMES.
    [literal-3  IS  mnemonic-name-1]
    [literal-4  IS  mnemonic-name-2] ...
    [GIN  IS  mnemonic-name-3]
    [SYSOUT  IS  mnemonic-name-4]
    [COMMUNICATION-DEVICE  IS  mnemonic-name-5]
    [REMOTE  IS  mnemonic-name-6]
    [GLAPS  IS  mnemonic-name-7]
    [GTIME  IS  mnemonic-name-8 [HMS]]
    [CONSOLE  IS  mnemonic-name-9]
    [TYPEWRITER  IS  mnemonic-name-10]
    [TOP  IS  mnemonic-name-11]
    [SWITCH integer-1  IS  mnemonic-name-12
      { , ON STATUS IS condition-name-1  [OFF STATUS IS condition-name-2] }
      { , OFF STATUS IS condition-name-2 [ON STATUS IS condition-name-1]  } ... ]
```

```
[BLOCK integer-2 IS data-name-1
   [THRU data-name-2]] ...
[COLLATE COMMERCIAL]
[OPTIMIZE  { COMPUTATIONAL }  ]
           { COMP          }
[DECIMAL-POINT IS COMMA]
[CURRENCY SIGN IS literal-5]
[PROCESS [{ ALL                                }  ]
         { LEVEL  integer-3 [THRU integer-4]   }
    DEBUG STATEMENTS]
[COMPILE PHASE1 ONLY [WITH SOURCE ERRORS]]
[ELECT SORT OPTIONS field-1,...,field-12
   FOR file-name-1 [, file-name-2] ...] ... .]
```

Syntax Rules:

1. The SPECIAL-NAMES paragraph must begin with the paragraph-name SPECIAL-NAMES which must be followed by a period and a space.

2. When Format 1 is used, the COPY library-name phrase is required. The 'library-name' must be identical to the name associated with the desired text on the library.

3. In Format 1, a word is any COBOL word.

4. When used as a report code, literal-3 or literal-4 must be a single-character nonnumeric literal whose value is a letter or a digit.

5. Mnemonic-names and condition-names may contain up to 30 characters, one of which must be a letter.

6. If mnemonic-names, condition-names, or compiler-directing phrases are used, the SPECIAL-NAMES paragraph must be included in the program.

7. File-name-1, file-name-2, ..., must be described in a sort-merge file description entry in the Data Division.

General Rules:

1. For a description of the COPY function, see Section VIII, the COBOL Library.

2. The literal IS mnemonic-name option is used to define report codes. This option is required when more than one report is to be generated by the program. The literal specified is the code appended to the report-name defined in the report description entry for the report. (Refer to the CODE clause.)

3. The 'special names' peculiar to the operating system (GIN, SYSOUT, COMMUNICATION-DEVICE, REMOTE, GLAPS, GTIME, CONSOLE, and TYPEWRITER) may be assigned mnemonic-names for use with the ACCEPT and DISPLAY statements. The mnemonic-names assigned these 'special names' may be referenced by the ACCEPT and DISPLAY statements as follows:

ACCEPT Statement	DISPLAY Statement
GIN	SYSOUT
COMMUNICATION-DEVICE	COMMUNICATION-DEVICE
REMOTE	REMOTE
GLAPS	CONSOLE
GTIME	TYPEWRITER
CONSOLE	
TYPEWRITER	

Although CONSOLE and TYPEWRITER are the same physical device, the effect of the ACCEPT and DISPLAY statements using these names is different. Refer to Section VI in the COBOL User's Guide.

4. If HMS (hours, minutes, seconds) is specified in the GTIME IS phrase, the receiving identifier in an ACCEPT statement must be described, explicitly or implicitly, with USAGE DISPLAY. If both forms of the GTIME phrase (with HMS and without HMS) are specified within the same COBOL program, the HMS option overrides and all GTIME requests are returned in hours, minutes, and seconds.

5. The mnemonic-name assigned to the special name 'TOP' may be referenced in a WRITE...ADVANCING statement to cause a listing to be advanced to the top of the next page.

6. The SWITCH option is used to associate user-specified mnemonic-names with the operating system's Program Switch Word switch numbers. When this option is used, at least one of the two possible states of the switch (ON or OFF) must be assigned a condition-name. The mnemonic-names associated with the software switches can be referenced only by using the ACCEPT and DISPLAY statements. The condition-names associated with the status of switches can be referenced only in sentences that contain either the IF or PERFORM statement. When the SWITCH option is used, the words SWITCH and IS are required as are either of the status names ON or OFF followed by IS.

7. The BLOCK option is used to allocate Labeled Common storage blocks in the Working-Storage Section. This option makes the contents of the labeled block accessible to programs containing a description of the labeled block. This option, therefore, facilitates program modularization. The label for each block is the integer specified by the user which can be either one or two digits. The data-names must be either level 01 or level 77 data items in the Working-Storage Section of the program and must not be REDEFINES data items. When the THRU phrase is used, data-name-2 must be in the same section as data-name-1. As many BLOCK sentences as required may be used; however, they must appear in the same order as the data items they refer to. A maximum of 63 Labeled Common storage blocks are permitted in a program.

8. The COLLATE COMMERCIAL option is a compiler-directing statement that is used when the Standard Collating Sequence of the machine is not wanted. When this option is specified, alphanumeric sort or merge keys and alphanumeric items in conditional statements of the type IF GREATER THAN... or IF LESS THAN..., are compared using the Commercial Collating Sequence. When this option is not used, the Standard Collating Sequence is used. For additional details, refer to the USAGE clause.

9. The OPTIMIZE COMPUTATIONAL or OPTIMIZE COMP option is available to improve object program efficiency. All elementary data items described with USAGE COMPUTATIONAL and which contain ten or less integral digits will be processed internally as single-precision binary integer data items if this option is specified.

   All elementary data items whose usage has been changed by utilizing this option will be flagged on the output source listing with the message 'WA--CONVERTED TO BINARY INTEGER'.

10. The DECIMAL-POINT IS COMMA option is a compiler-directing phrase that causes the function of the comma and the decimal point (period) to be exchanged in PICTURE clause character-strings, numeric literals, and object program editing.

11. The CURRENCY SIGN IS literal option is a compiler-directing phrase that substitutes the specified literal for the dollar sign in PICTURE character-strings and in object program editing. The literal specified must be a single character and must not be one of the following:

    a. Digits 0 through 9;

    b. Alphabetic characters A, B, C, D, E, J, K, P, R, S, V, X, Z; or the space;

    c. Special characters '*', '+', '-', ',', '.', ';', '(', ')', '"', '?', or '!'.

12. The PROCESS DEBUG STATEMENTS option is a compiler-directing phrase that allows processing of all or specific debugging statements in the source program. Debugging statements are identified by a single digit (0-9) in column 7 of the coding form. If it is desired to have all the debugging statements in the source program processed, the sentence PROCESS ALL DEBUG STATEMENTS is written. If only certain debugging statements are to be processed, the PROCESS LEVEL integer-3 DEBUG STATEMENTS phrase is written; then, only those debugging statements with the specified digit (integer-3) in column 7 are processed. In addition, a range of debugging statements may be processed by writing PROCESS LEVEL integer-3 THRU integer-4 DEBUG STATEMENTS; then, all debugging statements in the range specified (integer-3 THRU integer-4) are processed. When debugging statements identified by a single digit in column 7 appear in the source program and the PROCESS DEBUG STATEMENTS phrase is not included, those statements with a digit in column 7 are unconditionally bypassed (not processed). The PROCESS ALL DEBUG STATEMENTS option has no effect on statements that have a hyphen, a stroke, an asterisk, or a space in column 7.

13. The COMPILE PHASE1 ONLY option is a compiler-directing phrase used to optimize the compilation process by eliminating the output of an assembly listing and object program. A source listing with cross-references and error messages is produced at the end of phase 1 of the compilation and the compilation process stops. The COMPILE PHASE1 ONLY option may also be selected by specifying the COMPH1 option in the operand field of the $ COBOL card.

14. The COMPILE PHASE1 ONLY WITH SOURCE ERRORS option is a compiler-directing phrase used to optimize the compilation process by eliminating the output of an assembly listing and object program when source program errors are encountered. When this occurs, a source listing with cross-references and error messages is produced at the end of phase 1 of the compilation. If only warning and efficiency messages are generated during phase 1, the compilation process will go through to the end of phase 2 and a listing and object program will be produced. However, if an error is detected in phase 1, the compilation process stops at the end of phase 1.

15. The ELECT SORT OPTIONS phrase is used to exercise greater control over the sorting or merging process for sort or merge files (SD entries).

Twelve options (fields) which correspond to the ELECT macro as described for the freestanding sort/merge system are provided. (See the Sort/Merge Program manual.) The order of the fields must be preserved according to the numbers listed below. When the ELECT option is used, all fields must be specified according to field description or as a null field. In all cases, the first field must be specified. If file-name-2 is specified, it will receive the same options as file-name-1. No more than one ELECT may be given for a sort or merge file.

Field-1: Output Order Control

This parameter may be used to establish a tie-breaking convention for sort or merge key comparisons. Acceptable values are:

0 - The output order of equal key data records will be indeterminate.

1 - The original input order of the data will be used as the basis for breaking ties between equal keys. Selection will be made on the basis of original input sequence.

Field-2: Error Journal Control

This parameter overrides the standard journalization option in case of input or output errors. Acceptable values are:

0,N or null - Retain full journalization capabilities. (See the Sort/Merge Program manual.)

T - Journalize errors and then terminate processing through the abort routine of the operating system.

D - Do not journalize errors but continue processing. Although there is no journalization, an error occurrence message is placed on the execution report (SYSOUT).

A - Do not journalize errors. After placing an error occurrence message on the execution report, terminate processing through the abort routine of the operating system.

1 - Journalize errors. If the error occurred while reading the original input, continue processing. If the error occurred while reading a collation file, terminate processing through the abort routine of the operating system.

2 - Journalize errors. If the error occurred while reading the original input, continue processing. If the error was a block serial number error while reading a collation file, terminate processing through the abort routine of the operating system.

3 - Journalize errors. If the error was a block serial number error while reading the original input, terminate processing through the abort routine of the operating system. If the error occurred while reading a collation file, terminate processing through the abort routine of the operating system.

4 - Journalize errors. If the error was a block serial number error while reading either the original input or a collation file, terminate processing through the abort routine of the operating system.

The latter four options allow the user to discriminate between original input errors and collation file errors. They also allow different recovery options for general read errors and block serial number errors. This differentiation may be required because the journalization of a block serial error cannot contain the text of the missing block. If such an error occurs on a collation file, it is impossible to determine which records were dropped.

Field-3: Checkpoint Control

This parameter controls the taking of checkpoint records during a sort or merge process. Acceptable values are:

0 or
null - Do not take checkpoint records.

1 - Take checkpoint records. During the sort or merge process, checkpoints are written on whichever file is the current output file.

Field-4: Logical Record Memory Assignment Control

This parameter forces the first word of a data record into an even or odd memory location during a sorting process, and has no meaning for a merging process. This option is normally used to optimize comparison coding for double-precision keys. Acceptable parameter values are:

0 or
null - The placement of logical records in memory is indeterminate.

1 - The first word of every record being sorted is placed in an odd memory location.

2 - The first word of every record being sorted is placed in an even memory location.

Field-5: Borrow Tapes Control

This parameter allows the dynamic allocation of free tape handlers to a sort process at program execution. Such tapes are used for collation files in addition to the tapes allocated by $ TAPE or $ NTAPE cards. Acceptable parameter values are:

0 or
null - The sort process does not borrow tapes for collation files.

n - The n represents a numeric value from one to 13. The sort process borrows up to n tapes for collation files during execution. Borrowed tapes are released following the final collation pass.

Field-6: Input Device Positioning Control

This field allows the user to specify handling of the input file while opening and closing the file. The user is responsible for selecting appropriate disposition codes on file control cards. Acceptable values are:

	OPEN	CLOSE	
	Rewind	Rewind	Lock
0 or Null	Yes	Yes	No
1	Yes	Yes	Yes
2	Yes	No	No
3	Yes	Yes	No
4	Yes	Yes	No
5	No	Yes	Yes
6	No	No	No
7	No	Yes	No

Field-7: Output File Collation Control

This parameter controls the use of the output file as a collation file during a sorting process. Normally, the output file is so used. However, if the output file is not a tape device, this parameter must be used to inhibit its use as a collation file. Acceptable values are:

0 or
null - Use the output file as a collation working file.

1 - Do not use the output file as a collation working file.

Field-8: Output Device Positioning Control

This field allows the user to specify special handling of the output file while opening and closing the file. The user is responsible for selecting appropriate disposition codes on file control cards. Acceptable values are the same as those for Field-6.

Use of the values 5, 6, or 7 will prevent collation upon the output file. If one of these values is used and Field-7 is not one (1), this option will override Field-7 and an error message will be printed.

Field-9: Borrow Memory Control

This parameter allows the dynamic allocation of free memory to a sort process at program execution. The free memory area is used for control tables and buffers in addition to the area allocated by the $ LIMITS card.

This parameter is meaningful only in a $ LOWLOAD environment (see the Control Cards manual). Acceptable parameter values are:

0 or
null - No memory is borrowed for the sort process.

n - The n represents a numeric value of 1 to 262,144 memory locations. The sort process borrows n words of memory (or any available portion of n) at program execution. Memory is borrowed in 1024-word modules. All borrowed memory is released to the operating system following the final collation phase.

Field-10: Mass Storage FLR Mode Control

This parameter allows special processing of certain sort input files on linked mass storage. If the input file contains records which are all the same length, this parameter may be used to cause the sort process to operate in the fixed-length record mode, even though the records are in variable-length record format. This function does not apply to the merge process. Acceptable parameters are:

0 or
null - Assume normal processing.

1 - Engage special FLR processing. If this option is used, the sorting program must be set up as it would be for a FLR sort. The sort input must be specified through a USING phrase. Neither the input file, nor the sort file, nor the output file may be described with the phrases 'APPLY SYSTEM STANDARD' or 'APPLY VLR' in the I-O-CONTROL paragraph. None of the files may contain different sized multiple record descriptions. All of the record descriptions must be started with the item FILLER PIC X(6). This entry allows for the existence of the record control word which appears on each logical record. The comparison coding which the compiler generates is thereby properly aligned with the described key fields.

Field-11: Multiple Reel File Control (Optional)

This parameter forces automatic reel switching for labeled or unlabeled input files. If used, reel switching is forced at end-of-reel for both sort and merge input files. The end-of-input can be indicated only through operator intervention at reel switching time. Acceptable parameters are:

0 or
null - Assume normal processing.

1 - Assume input is on multiple reel unlabeled files.

2 - Assume input is on multiple reel labeled files.

Field-12: No Option Currently Implemented

INPUT-OUTPUT SECTION IN THE ENVIRONMENT DIVISION

The Input-Output Section in the Environment Division is optional. This section is subdivided into two paragraphs; the FILE-CONTROL paragraph and the I-O-CONTROL paragraph. If the Input-Output Section is included in a source program, the FILE-CONTROL paragraph header is required and the I-O-CONTROL paragraph is optional.

FILE-CONTROL Paragraph

The FILE-CONTROL paragraph names each file used in the program and identifies the type of peripheral device on which each file is stored. In addition, each file may be assigned to a particular input or output peripheral device. The paragraph can be written in any of the following three formats.

Format 1:

```
FILE-CONTROL.  COPY  library-name

[ REPLACING  word-1  BY { word-2      } ]
                        { literal-1   }
                        { identifier-1}

[ , word-3  BY { word-4      } ]... ].
               { literal-2   }
               { identifier-2}
```

Format 2:

```
FILE-CONTROL.

{ SELECT [ OPTIONAL ] [OVERLAY] file-name-1 [RENAMING file-name-2]

  ASSIGN TO [integer-1 ] file-code-1 [ , file-code-2 ] ...

        [ FOR { CARDS   } ]
              { LISTING }
              { MULTIPLE { REEL } } ]
                         { UNIT }

        [RESERVE { integer-2 } ALTERNATE [ AREA  ] ]
                 { NO        }           [ AREAS ]

        [FOR BLANK COMMON]
```

```
[,{FILE-LIMIT IS   } {data-name-1} {THRU    } {data-name-2}
  {FILE-LIMITS ARE} {literal-1   } {THROUGH} {literal-2   }
    [, {data-name-3} {THRU    } {data-name-4} ]...]
       {literal-3 } {THROUGH} {literal-4 }
[, ACCESS MODE IS {SEQUENTIAL}]
                  {RANDOM    }
[, PROCESSING MODE IS SEQUENTIAL]
[, ACTUAL KEY IS data-name-5] . }...
```

Format 3:

```
FILE-CONTROL.
 {SELECT file-name-1
  ASSIGN TO [ integer-1 ] file-code-1 [ , file-code-2] ... . }...
```

Syntax Rules:

1. The FILE-CONTROL paragraph is required when the INPUT-OUTPUT SECTION header is present.

2. The FILE-CONTROL paragraph must begin with the paragraph-name FILE-CONTROL which must be followed by a period and a space.

3. When Format 1 is used, the COPY library-name phrase is required. The 'library-name' must be identical with the name of the desired text on the library.

4. In Format 1, a word is any COBOL word.

5. The OPTIONAL phrase is allowed only for input files accessed in a sequential manner. It is required for sequential input files that are not necessarily present each time the object program is executed.

6. Each file described in the Data Division must be named once and only once as a file-name in the FILE-CONTROL paragraph following the keyword SELECT.

7. In Format 2, each selected file must have a file description entry in the Data Division.

8. In Format 3, each selected file must have a sort-merge file description entry in the Data Division. Sort files or merge files, those whose data descriptions begin with SD entries in the Data Division, must be named and assigned using Format 3.

9. Integer-1 and integer-2 must be unsigned nonzero integers.

10. Integer-1 is treated as documentation only since multiple devices are assigned using system control cards.

11. Integer-1 may not be specified when file-code-2, file-code-3, ..., is also specified in the ASSIGN phrase.

12. Multiple file-codes in the ASSIGN phrase are treated as documentation only.

13. File-code-1, ..., must be a two-character word consisting either of two letters (A through R, T through Z) or of one letter and one digit (0 through 9). File-codes beginning with the letter S should not be used in programs that utilize the sort or merge process. These file-codes, which include S1, S2, ..., SA, SB, ..., SZ, have a special meaning in the sort or merge operation. The code specified must not be a COBOL reserved word. The file-codes for each file named in a SELECT sentence must be unique within the program. When the object program is submitted for execution, it is accompanied by peripheral assignment cards which are used to specify the peripheral device for each file. The file-code in the peripheral assignment card must be the same as that assigned in the source program. Each of the object program's files is associated with the designated peripheral device when the operating system matches the file-codes.

14. The MULTIPLE REEL option must be specified whenever the number of magnetic tape devices might be less than the number of reels in the file.

15. The MULTIPLE UNIT option is treated as documentation only.

16. The RESERVE phrase allows the user to modify the number of input-output areas allocated by the compiler. The RESERVE integer ALTERNATE AREAS phrase indicates that the specified number of buffer areas are to be allocated in addition to the main buffer area reserved by the COBOL compiler. The value of integer-2 must be less than 256. If the RESERVE NO ALTERNATE AREAS phrase is specified, one buffer area is allocated. If the RESERVE phrase is omitted, the compiler automatically allocates one alternate buffer area in addition to the main buffer area for input-output.

17. The FILE-LIMIT(S) phrase is treated as documentation only. The FILE-LIMITS are determined by the amount of file space actually allocated to a particular file.

18. The ACCESS MODE and PROCESSING MODE phrases must be given for mass storage files.

General Rules:

1. For a description of the COPY function, see Section VIII, the COBOL Library.

2. For files common to more than one program, the OVERLAY phrase is used to indicate that the control information for the file is to be used in an overlay environment. The overlay area is constructed so that the file can be overlayed and then be referred to as it was prior to being overlayed. For example, a file can be opened, then overlayed, and then referred to without first closing it and then opening it again. To accomplish this overlaying feature, the OVERLAY phrase must be specified in a SELECT sentence for each program in which this file is common, except for that program that initially references the file. Note that an 'overlay file' does not generate a 'normal' file control block. Rather, the location symbol (or symbolic address of the file control information) is positioned within the correct Labeled Common storage area by the compiler. The file properties (including any RERUN phrases) of all files using this feature must be identical.

3. The file-name supplied as file-name-1 in each SELECT sentence must be exactly the same as the file-name that appears in the file description entry or sort-merge file description entry in the Data Division, except when the RENAMING phrase is used. When RENAMING is used, the data description entry associated with file-name-2 is applied to the file-name specified as file-name-1 in this SELECT sentence. The data description entry associated with file-name-2 includes its file description entry and associated record description entries. Because of this, the file-name specified as file-name-1 must not be described in the File Section of the Data Division. The file description for the renamed file must not be the last file description in the File Section. Also, the SELECT sentence for the file-name specified as file-name-2 must not contain a RENAMING phrase. The renamed file must not have a sort-merge file description. The use of the RENAMING phrase does not imply the use of the SAME AREA phrase of the I-O-CONTROL paragraph. Each file named in each SELECT sentence must have a unique name. When the RENAMING phrase is used, the COPY option on the $ COBOL card must also be used and the LIBCPY option must not be used.

4. Each file named in a SELECT sentence must be assigned to a peripheral device using the ASSIGN TO phrase.

5. Normally, space is allocated in the Labeled Common storage area for input-output buffer areas. However, if the FOR BLANK COMMON phrase is specified, the buffer space for the file is allocated within Blank Common storage rather than in Labeled Common storage. This feature is useful when loading large object programs when a limited amount of memory is available. In this case, the Blank Common area can be shared with the loader by including the variable-field option on a $ LOWLOAD card. If the $ LOWLOAD option is used, the maximum size of Blank Common must be specified on the $ LOWLOAD card. The octal length of the Blank Common storage area is printed on the preface page of each GMAP listing. Refer to the description of the $ LOWLOAD card in the General Loader reference manual.

The FOR BLANK COMMON phrase cannot be used in any program that requires segmentation.

The FOR BLANK COMMON phrase can be used with independent programs without creating problems, but caution is advised when using this feature with programs in a module overlay environment in which files using this feature may be common to another program. If a file common to two or more programs is to be assigned to Blank Common storage, the file must be so assigned in each program in which it is common. An identical ordering of files (by SELECT sentences) in each program is required when more than one such file is involved.

6. The ACCESS MODE IS phrase should not be used for non-mass-storage files but is required for mass storage files. When ACCESS MODE IS SEQUENTIAL is specified, the mass storage logical records are read or written sequentially. The user need not specify the actual key (through the ACTUAL KEY IS phrase) when the access mode of a mass storage file is sequential. When ACCESS MODE IS RANDOM is specified, the actual key must be specified in the ACTUAL KEY IS phrase. When the access mode is random, the mass storage logical records are read and written randomly using the data-name contents of the actual key to locate or place the records.

7. The PROCESSING MODE IS phrase has no meaning for non-mass-storage files but is required for mass storage files. When PROCESSING MODE IS SEQUENTIAL is specified, the mass storage logical records are processed in the order in which they are accessed. Thus, if logical records are accessed randomly, the records are processed in the order of access.

8. The ACTUAL KEY IS phrase has no meaning for non-mass-storage files. For mass storage files, the ACTUAL KEY phrase is optional if access is sequential and is required if access is random. The ACTUAL KEY, given as data-name-5, must be a single-precision binary integer with usage described as USAGE COMP-1. Furthermore, the ACTUAL KEY must be described in the Data Division as either a level 01 or level 77 entry in the Working-Storage Section. When the access mode is random, the user can obtain records sequentially by incrementing by one the value of the data-name associated with ACTUAL KEY.

Special Considerations:

1. When a file is assigned to an implementor-name that implies a card image media code (FOR CARDS) or a print line media code (FOR LISTING), the second character of the first file-code assigned to that file is used as the report code for that file when a WRITE or WRITE...ADVANCING statement is executed. If multiple report files are assigned to the same physical device, report segregation may be accomplished by assigning file-codes that contain unique second characters.

   Example:

```
FILE-CONTROL.
    SELECT  REPORT-1  ASSIGN  TO  AA  FOR  LISTING.
    SELECT  REPORT-2  ASSIGN  TO  AB  FOR  LISTING.
    SELECT  REPORT-3  ASSIGN  TO  AC  FOR  CARDS.

PROCEDURE DIVISION.
    WRITE  RECORD-NAME-1.       (report code A)
    WRITE  RECORD-NAME-2  ...  ADVANCING.  (report code B)
    WRITE  RECORD-NAME-3.       (report code C)
```

   In this example, unique report codes would be affixed to REPORT-1, REPORT-2, and REPORT-3, thus allowing report segregation via SYSOUT or utility routines. However, in the example

```
FILE-CONTROL.
    SELECT  REPORT-1  ASSIGN  TO  AA  FOR  LISTING.
    SELECT  REPORT-2  ASSIGN  TO  BA  FOR  LISTING.
```

   the identical WRITE sequence specified above would affix the report code 'A' to both reports and intermingled output would be produced.

I-O-CONTROL Paragraph

The I-O-CONTROL paragraph defines special control techniques to be used in the object program. Input-output techniques, the points at which rerun is to be established, the memory area that is to be shared by different files, and the location of files on a multiple file reel can be specified in the I-O-CONTROL paragraph. The paragraph can be written in either of the following two formats.

Format 1:

```
I-O-CONTROL.  COPY  library-name

[REPLACING  word-1  BY  {word-2      }
                        {literal-1   }
                        {identifier-1}

[, word-3  BY  {word-4      }]  ...].
               {literal-2   }
               {identifier-2}
```

Format 2:

```
I-O-CONTROL.

[APPLY PROCESS AREA ON file-name-1 [, file-name-2] ...]

[APPLY BLOCK SERIAL NUMBER ON file-name-3[ , file-name-4] ...]

[APPLY {SYSTEM STANDARD} FORMAT ON file-name-5
       {VLR             }

       [, file-name-6] ...]

[RERUN [ON file-name-7]

       EVERY integer-1 RECORDS OF file-name-8] ...
```

```
[ SAME [ { RECORD      } ]  AREA FOR file-name-9, file-name-10
       [ { SORT-MERGE } ]
       [ { SORT       } ]

          [, file-name-11, file-name-12] ... ] ...

[ MULTIPLE FILE TAPE CONTAINS file-name-13 [POSITION integer-2]

          [, file-name-14 [POSITION integer-3] ...] ] ...
```

Syntax Rules:

1. The I-O-CONTROL paragraph is optional. If present, it must begin with the paragraph-name I-O-CONTROL which must be followed by a period and a space.

2. When Format 1 is used, the COPY library-name phrase is required. The 'library-name' must be identical to the name associated with the desired text on the library.

3. In Format 1, a word is any COBOL word.

4. APPLY phrases may be separated by a semicolon.

5. A file-name that represents a sort file or merge file cannot appear in a RERUN phrase or a MULTIPLE FILE phrase.

6. In the SAME AREA phrase, SORT and SORT-MERGE are equivalent.

7. A file-name that represents a sort file or merge file must not appear in the SAME sentence unless the SORT, SORT-MERGE, or RECORD option is used.

8. The four forms of the SAME phrase (SAME AREA, SAME RECORD AREA, SAME SORT AREA, SAME SORT-MERGE AREA) are considered separately in the following:

   More than one SAME sentence may be included in a program. However:

   a. A file-name must not appear in more than one SAME AREA phrase.

   b. A file-name must not appear in more than one SAME RECORD AREA phrase.

   c. A file-name that represents a sort file or merge file must not appear in more than one SAME SORT AREA or SAME SORT-MERGE AREA phrase.

d. If one or more file-names of a SAME AREA phrase appear in a SAME RECORD AREA phrase, all of the file-names in that SAME AREA phrase must appear in that SAME RECORD AREA phrase. However, additional file-names not appearing in that SAME AREA phrase may also appear in that SAME RECORD AREA phrase. The rule that only one of the files mentioned in a SAME AREA phrase can be open at any given time takes precedence over the rule that all files mentioned in a SAME RECORD AREA phrase can be open at any given time.

e. If a file-name that does not represent a sort file or merge file appears in a SAME AREA phrase and one or more SAME SORT AREA or SAME SORT-MERGE AREA phrases, all of the files named in that SAME AREA phrase must be named in that SAME SORT AREA or SAME SORT-MERGE AREA phrase(s).

9. Each file specified in the MULTIPLE FILE phrase must be named in a SELECT sentence.

General Rules:

1. For a description of the COPY function, see Section VIII, the COBOL Library.

2. The APPLY PROCESS AREA ON phrase is provided to direct the compiler to generate a fixed logical record processing area (PROCESS AREA) in addition to the normal buffer area(s). (See the RESERVE ALTERNATE AREA phrase in the FILE-CONTROL paragraph.) When the APPLY PROCESS AREA phrase is included, each logical record of an input file is moved to the 'PROCESS AREA' for processing when it is read and each logical record of an output file is developed in the 'PROCESS AREA' and moved to the buffer when it is written. Otherwise, the standard method is to process both input and output files in the buffer area.

   Data descriptions for files that utilize a process area, either explicitly or implicitly, must contain record description entries for the largest records that will be encountered on the file or device. For example, the data description for a card file must have a record description that describes all of the columns of the card medium.

3. The APPLY BLOCK SERIAL NUMBER ON phrase is provided for specifying to the compiler those files that are physically blocked with a serial number. All blocked files with serial numbers must be specified in an APPLY BLOCK SERIAL NUMBER phrase because the input routine (as an I-O error control feature) checks the block serial numbers of the files when they are read. If such a file is not specified in an APPLY BLOCK SERIAL NUMBER phrase or implicitly given serial numbers (see below), an error condition will result when that file is read.

I-O-CONTROL

4. The APPLY SYSTEM STANDARD FORMAT ON phrase is provided as a shorthand method of describing certain physical aspects of a file. When the SYSTEM STANDARD FORMAT phrase is specified:

   a. Block serial numbers are applied.

   b. The data block size will be 320 words (1920 BCD characters).

   c. Variable-length records (VLR) will be assumed; that is, each logical record will be preceded in the buffer by a record control word containing the record size and other information.

d. The recording mode will be BINARY HIGH DENSITY.

e. Label records will be STANDARD.

If the APPLY SYSTEM STANDARD FORMAT phrase is not used, these standards may be specified in the appropriate clause(s) in the FD entry for the file. If the APPLY SYSTEM STANDARD FORMAT phrase is used and the RECORDING MODE, BLOCK CONTAINS, RECORD CONTAINS, or LABEL clauses are also specified in the FD entry for the file, there must be no deviation from the system standard format in the FD entry clauses.

5. When the APPLY VLR FORMAT phrase is specified, the logical records of the file are preceded in the buffer by a record control word that contains the record size (in words) and other control information. Also, the APPLY VLR phrase implicitly specifies that the recording mode is binary. Depending on what is specified in the BLOCK CONTAINS, RECORD CONTAINS, and LABEL clauses of the FD entry for the file, the APPLY VLR FORMAT specification may or may not apply to a file that conforms to the system standard format.

6. The RERUN phrase is used to cause checkpoint memory dumps to be written. If 'ON file-name-7' is specified, the output device allocated to file-name-7 receives the checkpoint dump; otherwise, the output device allocated to file-name-8 receives the checkpoint dump. If 'ON file-name-7' is specified, file-name-8 may be either an input or an output file. The number of records specified by integer-1 may not exceed 250,000. The output device must be opened as an output file at every point in the program where a READ or a WRITE statement references file-name-8 so that the output device can receive the checkpoint dump.

7. The SAME AREA phrase specifies that two or more files that are not sort files or merge files are to use the same memory area during processing. The area to be shared includes all storage areas (and alternate areas) assigned to the files specified. It is not valid, therefore, to have more than one of the files open at the same time. A file-name must not be used in more than one SAME AREA phrase.

8. The SAME RECORD AREA phrase specifies that two or more files are to use the same memory area for processing the current logical record. If the files named in the SAME RECORD AREA phrase are not also named in a SAME AREA phrase, then all the files named in the SAME RECORD AREA phrase can be open at the same time. A logical record that is processed in the 'Same Record Area' is considered to be a logical record of each opened output file that is named in this SAME RECORD AREA phrase. In addition, a logical record that is processed in the 'Same Record Area' is considered to be a logical record of the most recently read input file that is named in this SAME RECORD AREA phrase. A file may be specified in only one SAME RECORD AREA phrase.

9. If the SAME SORT AREA phrase or the SAME SORT-MERGE AREA phrase is used, at least one of the file-names must represent a sort file or merge file. Files that do not represent sort files or merge files may also be named in the phrase. Storage is shared as follows:

The SAME SORT AREA or SAME SORT-MERGE AREA phrase specifies a memory area which will be made available for use in sorting or merging each sort file or merge file named. Thus, any memory area allocated for the sorting or merging of a sort or merge file is available for reuse in sorting or merging any of the other sort or merge files.

In addition, memory areas assigned to files that do not represent sort files or merge files may be allocated as needed for sorting or merging the sort files or merge files named in the SAME SORT AREA or SAME SORT-MERGE AREA phrase.

Files other than sort files or merge files do not share the same memory area with each other. If a user wishes these files to share the same memory area with each other, a SAME AREA or SAME RECORD AREA phrase naming these files must also be included in the program.

During the execution of a SORT or MERGE statement that refers to a sort file or merge file named in the SAME SORT AREA or SAME SORT-MERGE AREA phrases, any non-sort-merge files named in these phrases must not be open.

10. The MULTIPLE FILE phrase is required when two or more files share the same reel of tape. Only those files on a multiple file tape that are referenced elsewhere in the source program need be named in a MULTIPLE FILE phrase. If all file-names on the tape are listed consecutively, the POSITION option may be omitted. If any file in the sequence of files on the tape is not included in the MULTIPLE FILE phrase, then the position relative to the beginning of the tape of each file named in the phrase must be given.

    All the files on a multiple file tape must either have labels present or have labels omitted. Each MULTIPLE FILE phrase describes one multiple file tape. There can be any number of multiple file input or output tapes (each having a corresponding MULTIPLE FILE phrase); however, all files listed for each tape must be contained on a single reel. Only one file of a multiple file tape can be open at any given time. Note that OPTIONAL files (those specified as such in a SELECT sentence in the FILE-CONTROL paragraph) are not permitted on multiple file tapes.

Special Considerations:

1. The APPLY PROCESS AREA phrase can be used to increase object program efficiency on blocked or buffered files that have heavy processing activity.

2. The APPLY BLOCK SERIAL NUMBER phrase has no meaning for sort files and merge files (which are always blocked with block serial numbers).

3. At program execution, the unique file-codes associated with the files listed in a given MULTIPLE FILE phrase must be equated to the same logical unit designator using the appropriate $ TAPE file control cards. *

SECTION VI

DATA DIVISION

DESCRIPTION OF THE DATA DIVISION

The third division of a COBOL source program is the Data Division and it is required. It is used to describe data that the object program is (1) to accept as input, (2) to manipulate, (3) to create, or (4) to produce as output. Data to be processed can be that which is contained in files and enters or leaves the internal memory of the computer from a specified area or areas, data which is developed internally and placed into intermediate storage or working-storage or placed into specific format for output reporting purposes, or constants which are defined by the user.

Organization of the Data Division

The Data Division is subdivided into three sections; the File Section, the Working-Storage Section, and the Report Section.

The File Section is used to define the contents of data files stored on external media. Each file is defined by a file description entry followed by a record description or a series of record descriptions.

The Working-Storage Section is used to describe records and noncontiguous data items which are not part of external data files but are developed and processed internally. It also describes data items whose values are assigned in the source program and do not change during the execution of the object program.

The Report Section is used to describe the content and format of reports that are to be generated.

Structure of the Data Division

The Data Division is prepared in accordance with the reference format described in Section III. The Data Division is identified by and must begin with the division header DATA DIVISION followed by a period and a space. The sections of the Data Division are optional only if the functions filled by each are not required to describe the data for the object program. The general format of the Data Division is given below.

General Format:

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

REPORT SECTION.

The names of the sections in the Data Division are fixed and their required order of appearance is as shown. The section header for the File Section is followed by one or more sets of entries composed of file description entries or sort-merge file description entries which are followed by associated record description entries. The section header for the Report Section is followed by one or more sets of entries composed of report description entries which are followed by associated report group description entries. The Working-Storage Section header is followed by data description entries for noncontiguous items, if any, followed by record description entries.

STRUCTURE OF A RECORD DESCRIPTION

A record description consists of a set of data description entries that describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name (if required), followed by a series of independent clauses as required. A record description has a hierarchical structure; therefore, the clauses used with an entry may vary considerably, depending on whether or not the entry is followed by subordinate entries. The record description structure is further defined in the Concept of Levels paragraph in Section III. The elements allowed in a record description are shown in the data description skeleton which follows in this section.

FILE SECTION IN THE DATA DIVISION

The File Section contains file descriptions (FD entries), sort file or merge file descriptions (SD entries), and record descriptions (level-number entries) for both label and data records in files and for data records in sort files. Label records and data records are defined in the same manner but fixed names have been assigned to certain label record items to permit the input-output system to perform special operations on certain items of label records. All record description entries pertaining to label records and data records of a file must immediately follow the FD entry for the file. The FD and SD entries represent the highest level of organization in the File Section.

The File Section header is followed by a file description entry or a sort-merge file description entry. A file description entry consists of a level indicator (FD), followed by a data-name (the name of the file) and a series of independent clauses. The clauses specify:

- The manner in which data is recorded on the file.
- The size of the logical and physical records.
- The names of the label records contained in the file and values of label items.

- The names of the data records of which the file is composed.

The entry is terminated with a period.

For sort or merge file descriptions, the level indicator SD is followed by a data-name (the name of the file) and a series of independent clauses. The clauses specify the name, size, and number of data records in the sort or merge file. Note that a sort file is a set of records to be sorted or merged using a SORT or MERGE statement (in the Procedure Division). Therefore, no label procedures are under the control of the user and the rules for blocking and internal storage are peculiar to the SORT or MERGE statement.

WORKING-STORAGE SECTION IN THE DATA DIVISION

Working-storage is that part of memory set aside for the intermediate processing of data. The difference between working-storage and file storage is that working-storage concerns the memory requirements for the storage of intermediate data results while file storage concerns the memory requirements for the storage of each record of the file.

The Working-Storage Section consists of the section header, followed by data description entries for noncontiguous working-storage data items and record description entries, in that order. Each Working-Storage Section record-name and each noncontiguous item name must be unique since they cannot be qualified. Subordinate data-names need not be unique if they can be made unique through qualification.

The initial value of any data item in the Working-Storage Section except an index data item is specified by using the VALUE clause with the data item. VALUE can be specified only in terms of homogeneous characters (characters having the same usage). Therefore, VALUE cannot be specified in a group item containing elementary items that have different usages. All of the rules for the expression of literals and figurative constants apply and the use of the VALUE clause cannot contravene these rules. The size of a literal used to specify an initial value of an alphabetic or alphanumeric item can be equal to or less than the size specified in the PICTURE clause of the associated data entry, but the size of the literal cannot be greater than that. When the size of the literal is less, the normal rules for a MOVE statement for the literal apply. The size of a literal used to specify an initial value of a numeric item may be greater than the size specified in the PICTURE clause of the associated data entry, but the literal must not have a value that would require the truncation of nonzero digits. The initial value of index data items in working-storage cannot be predicted.

Noncontiguous Working-Storage

Items and constants in working-storage that are not related to one another in a hierarchy need not be grouped in records if they require no further subdivision. Instead, in working-storage, they are classified (and defined by the user) as noncontiguous elementary items. Each noncontiguous elementary item is defined in a separate data description that begins with the special level-number, 77. For each level 77 data description entry, the data-name of the item must be specified and a PICTURE clause must be supplied. Other data description clauses are optional and can be used to complete the description of the item if necessary. However, the OCCURS clause is not meaningful in level 77 entries and will cause a compilation error if used.

Working-Storage Records

Data elements and constants in working-storage that are related to one another in a hierarchy must be grouped into records according to the rules for the formation of record descriptions (data description entries). All of the clauses used in a data description entry including REDEFINES, OCCURS, and COPY may be used in a working-storage record description. The skeletal format of the Working-Storage Section is given below.

```
WORKING-STORAGE SECTION.

77 data-description entry
   88 condition-name-1
  .
  .
  .
77 data-description entry
01 data-description entry
   02 data-description entry
  .
  .
  .
66 data-name-n RENAMES data-name-m
01 data-description entry
   02 data-description entry
      03 data-description entry
         88 condition-name-2
```

REPORT SECTION IN THE DATA DIVISION

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

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

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

The Report Section consists of two types of entries for each report; one describes the physical aspects of the report format, and the other describes conceptual characteristics of the items which make up the report and their relation to the report format. These are:

- Report description entry (RD).
- Report group description entries.

File Description - Complete Entry Skeleton

The file description specifies information concerning the physical structure identification, and record-names that apply to a given file. The general formats of the FD entry, syntax rules for the complete entry, and special considerations for the entry follow. The individual clauses are described later in this section.

Format 1:

```
FD file-name  COPY  library-name
    [ REPLACING  word-1  BY { word-2 | literal-1 | identifier-1 }
        [ , word-3  BY { word-4 | literal-2 | identifier-2 } ] ... ] .
```

Format 2:

```
FD file-name

    [ BLOCK CONTAINS [integer-1 TO] integer-2 { RECORDS | CHARACTERS } ]

    [ DATA { RECORD IS | RECORDS ARE } data-name-1 [, data-name-2] ... ]

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

    [ RECORD CONTAINS [integer-3 TO] integer-4  CHARACTERS ]

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

    [ { REPORT IS | REPORTS ARE } report-name-1 [, report-name-2] ... ]
```

```
[VALUE OF data-name-5 IS  { literal-1      }
                          { data-name-6▲   }

    [ , data-name-7 IS    { literal-2      } ] ... ] .
                          { data-name-8▲   }
```

Syntax Rules:

1. The level indicator FD identifies the beginning of a file description and must precede the file-name.
2. The clauses following the file-name in Format 2, except for the LABEL RECORD(S) clause, are optional and the order of entry is not significant.
3. The FD entry must be terminated by a period.
4. The DATA RECORD(S) clause and the REPORT(S) clause must not both appear in the same file description entry.

General Rules:

1. For a description of the COPY function, see Section VIII, the COBOL Library.
2. Format 1 is used when the COBOL library contains the file description entry; otherwise, Format 2 is used.

Sort-Merge File Description - Complete Entry Skeleton

The sort-merge file description specifies information concerning the physical structure, identification, and record-names of the file to be sorted or merged. The general formats of the SD entry, syntax rules for the complete entry, and special considerations for the entry follow. The individual clauses are described later in this section.

Format 1:

```
SD file-name  COPY  library-name

    [ REPLACING  word-1  BY { word-2      }
                            { literal-1   }
                            { identifier-1}

-       [ , word-3  BY { word-4      } ]  ...  ].
                       { literal-2   }
                       { identifier-2}
```

Format 2:

```
SD file-name

    [ DATA { RECORD IS  } data-name-1[  , data-name-2] ...]
           { RECORDS ARE}

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

Syntax Rules:

1. The level indicator SD identifies the beginning of the sort-merge file description and must precede the file-name.
2. The clauses following file-name in Format 2 are optional and their order of entry is at the discretion of the user.
3. The SD entry must be terminated by a period.

SD SKELETON

General Rules:

1. For a description of the COPY function, see Section VIII, the COBOL Library.

2. Format 1 is used when the COBOL library contains the sort-merge file description entry; otherwise, Format 2 is used.

3. The file-name used in an SD entry can be referenced only in SORT, MERGE, and RETURN statements of the Procedure Division, except when it is used as a qualifier.

Report Description - Complete Entry Skeleton

The report description entry contains information pertaining to the overall format of a report named in the File Section and is uniquely identified in the Report Section by the level indicator RD. The characteristics of the report page are provided by describing the number of physical lines per page and the limits for presenting specified headings, footings, and details within a page structure. Data items which act as control factors during presentation of the report are specified in the RD entry. Each report named in an FD entry in the File Section must be defined by an RD entry. The general formats of the RD entry and syntax rules follow. The individual clauses are described later in this section.

Format 1:

```
RD report-name [ CODE mnemonic-name ] COPY library-name

[ REPLACING word-1 BY { word-2       }
                      { literal-1    }
                      { identifier-1 }

    [ , word-3 BY { word-4       } ]... ] .
                  { literal-2    }
                  { identifier-2 }
```

Format 2:

```
RD report-name

  [ CODE mnemonic-name-1 ]

  [ { CONTROL IS  } { FINAL                                          }
    { CONTROLS ARE } { identifier-1 [ , identifier-2 ] ...           }
                    { FINAL , identifier-1 [ , identifier-2 ] ... } ]

  [ PAGE { LIMIT IS  } integer-1 { LINE  }
         { LIMITS ARE }           { LINES }

      [ , HEADING integer-2 ]
      [ , FIRST DETAIL integer-3 ]
      [ , LAST DETAIL integer-4 ]
      [ , FOOTING integer-5 ] ] .
```

Syntax Rules:

1. The level indicator RD identifies the beginning of a report description entry and must precede the report-name.

2. The report-name must appear in at least one FD entry REPORT(S) clause.

3. The clauses following the report-name, except for the COPY clause in Format 1, are optional. The clauses may be defined in any order except for the CODE clause which, if specified, must immediately follow the report-name.

4. The RD entry must be terminated by a period.

General Rules:

1. For a description of the COPY function, see Section VIII, the COBOL Library.

2. Format 1 is used when the COBOL library contains the report description entry; otherwise, Format 2 is used. If the library-name is not unique, it may be qualified.

3. The reserved words LINE-COUNTER and PAGE-COUNTER are automatically generated by the Report Writer based on specific entries and are not data clauses. Refer to the PAGE LIMIT(S) clause for additional information.

DATA DESCRIPTION
SKELETON

Data Description - Complete Entry Skeleton

The data description entry is used to provide information concerning the characteristics of a particular item of data. 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 detailed data descriptions in COBOL involve a hierarchical structure, the contents of an entry may vary considerably, depending on whether or not the entry is followed by subordinate entries. In defining the lowest level or subdivision of data, the following information may be required:

1. A level-number showing the relationship between this and other units of data.
2. A data-name.
3. The predominant usage of the item (COMPUTATIONAL or DISPLAY).
4. The number of consecutive occurrences of elements in a table.
5. The type of data item being described (alphabetic, numeric, or alphanumeric).
6. The presence of an operational sign (+ or -).
7. The location of an actual or assumed decimal point.
8. The location of editing symbols (such as $ and ,).
9. Justification and synchronization of the data.
10. Special editing requirements such as zero suppression.
11. The initial value of an item or fixed value of a constant in working-storage.

An entry defining a unit of data must not be contradicted by a subordinate entry. For example, after USAGE is defined, it applies to all subordinate entries and need not be respecified in the subordinate entries.

The general formats of the detailed data description entry and syntax rules follow. The individual clauses are described later in this section.

DATA DESCRIPTION
SKELETON

Format 1:

```
01  data-name  COPY  library-name
    [ REPLACING  word-1  BY { word-2      }
                            { literal-1   }
                            { identifier-1}
      [ , word-3  BY { word-4      } ] ... ] .
                     { literal-2   }
                     { identifier-2}
```

Format 2:

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

Format 3:

```
level-number { data-name-1 }
             { FILLER      }

  [ REDEFINES data-name-2 ]

  [ { PICTURE } IS character-string ]
    { PIC     }

  [ [USAGE IS] { COMPUTATIONAL   }
               { COMP            }
               { COMPUTATIONAL-1 }
               { COMP-1          }
               { COMPUTATIONAL-2 }
               { COMP-2          }
               { COMPUTATIONAL-3 }
               { COMP-3          }
               { COMPUTATIONAL-4 }
               { COMP-4          }
               { DISPLAY         }
               { DISPLAY-1       }
               { DISPLAY-2       }
               { INDEX           } ]
```

DATA DESCRIPTION
SKELETON

```
[ OCCURS { integer-1  TO  integer-2  TIMES [ DEPENDING ON data-name-3 ] }
         { integer-2  TIMES                                            }

    [ { ASCENDING  } KEY IS data-name-4  [ , data-name-5 ] ... ] ...
      { DESCENDING }

    [ INDEXED BY index-name-1 [ , index-name-2 ] ... ] ]

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

[ { JUSTIFIED } RIGHT ]
  { JUST      }

[ BLANK WHEN ZERO ]

[ VALUE IS literal-1 ] .
```

Format 4:

```
66 data-name-1  RENAMES data-name-2

    [ { THRU      } data-name-3 ] .
      { THROUGH   }
```

Format 5:

```
88 condition-name { VALUE IS  } literal-1
                  { VALUES ARE }

    [ { THRU    } literal-2 ]
      { THROUGH }

  [ , literal-3  [ { THRU    } literal-4 ] ] ... .
                   { THROUGH }
```

DATA DESCRIPTION
SKELETON

Syntax Rules:

1. A data description entry must be terminated by a period.

2. In Format 3, the level-number can be any number in the range 01-49 or it can be 77.

3. The clauses can be written in any order except that:

   a. The data-name-1 or FILLER clause must immediately follow the level-number.

   b. The REDEFINES clause, when used, must immediately follow the data-name-1 clause.

4. The PICTURE clause must be specified for every elementary data item except an index data item, in which case use of this clause is prohibited.

5. The words THRU and THROUGH are equivalent.

General Rules:

1. For a description of the COPY function (Formats 1 and 2), see Section VIII, the COBOL Library.

2. The SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO clauses must only be specified for elementary data items.

3. Format 5 is used for each condition-name. Each condition-name requires a separate level-number 88 entry. Format 5 specifies the name of the condition and the value, values, or range of values associated with the condition-name. The condition-name entries for a particular conditional variable must follow the entry describing the item with which the condition-name is associated. A condition-name can be associated with any data description entry that contains a level-number except for the following:

   a. Another condition-name.

   b. A level 66 item.

   c. A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE DISPLAY).

   d. An index data item.

REPORT GROUP
SKELETON

Report Group Description - Complete Entry Skeleton

A report group may be as complex as a set of data made up of several print lines with many data items or as simple as one print line with one data item. A description of a set of data becomes a report group by the presence of a level-number 01 and a TYPE description. The level-number gives the depth of the group and the TYPE describes the purpose of the report group presentation. At object program execution, report groups are created as a result of Report Writer GENERATE statements in the Procedure Division.

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

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

The general formats of the report group description entry and syntax rules follow. The individual clauses are described later in this section.

Format 1:

```
01  [data-name]  COPY library-name

    [ REPLACING   word-1   BY  { word-2      }
                               { literal-1    }
                               { identifier-1 }

        [ , word-3   BY { word-4      } ]   ... ] .
                        { literal-2   }
                        { identifier-2}
```

REPORT GROUP
SKELETON

Format 2:

```
01  [data-name-1]

    [ LINE NUMBER IS { integer-1          } ]
                     { PLUS integer-2     }
                     { NEXT PAGE          }

    [ NEXT GROUP IS  { integer-3          } ]
                     { PLUS integer-4     }
                     { NEXT PAGE          }

                     {  REPORT HEADING                                    }
                     {  RH                                                }
                     {  PAGE HEADING                                      }
                     {  PH                                                }
                     {  OVERFLOW HEADING                                  }
                     {  OH                                                }
                     { {CONTROL HEADING} }   { identifier-1 }             }
    TYPE IS          { {CH              }   { FINAL         }             }
                     {  DETAIL                                            }
                     {  DE                                                }
                     { {CONTROL FOOTING} }   { identifier-2 }             }
                     { {CF              }   { FINAL         }             }
                     {  OVERFLOW FOOTING                                  }
                     {  OV                                                }
                     {  PAGE FOOTING                                      }
                     {  PF                                                }
                     {  REPORT FOOTING                                    }
                     {  RF                                                }

    [[USAGE IS]  { DISPLAY   } ] .
                 { DISPLAY-1 }
```

REPORT GROUP
SKELETON

Format 3:

```
level-number [ data-name-1 ]

    [ COLUMN NUMBER IS integer-1 ]

    [ BLANK WHEN ZERO ]

    [ GROUP INDICATE ]

    [ { JUSTIFIED } RIGHT ]
      { JUST      }

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

    [ { PICTURE } IS character-string ]
      { PIC     }

    [ RESET ON { identifier-1 } ]
               { FINAL        }

    { SOURCE IS [ SELECTED ] identifier-2                                  }
    { SUM identifier-3 [, identifier-4] ... [ UPON data-name-2 ]           }
    { VALUE IS literal-1                                                   }

    [ [ USAGE IS ] { DISPLAY   } ] .
                   { DISPLAY-1 }
```

Format 4:

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

REPORT GROUP
SKELETON

Syntax Rules:

1. The clauses in Format 1 must be presented in the order shown. Except for the data-name clause, which must immediately follow the level-number when present, the clauses in Formats 2 and 3 may be written in any order.

2. A report group must have a data-name in order to be referred to by a Procedure Division statement.

3. If the COLUMN NUMBER clause is present in the data description of an elementary data item, the data description must also contain the PICTURE clause and one of the SOURCE, SUM, or VALUE clauses.

4. The report group description entry must be terminated by a period.

5. In Format 3 and Format 4, level-number may be any number from 01 to 49.

General Rules:

1. For a description of the COPY function (Formats 1 and 4), see Section VIII, the COBOL Library.

2. Format 2 is used to indicate a report group; the report group extends from this entry to the next report group level 01 entry.

3. When LINE NUMBER is specified in Format 2, entries for the first report line within the report group are presented on the specified line.

4. Format 3 is used to indicate an elementary item or group item within a report group.

5. If a report group is an elementary entry, Format 3 may include the TYPE and NEXT GROUP clauses to specify the report group and elementary item in the same entry.

6. When LINE NUMBER is specified for Format 3, sequential entries with the same level-number in the report group are implicitly presented on the same line.

7. A LINE NUMBER at a subordinate level must not contradict a line number at a group level.

8. The NEXT GROUP clause, when specified, refers to the spacing (at object program execution) between the last line of this report group and the first line of the next report group.

9. Variable-length items may not be defined in the Report Section.

Descriptions of the Data Division clauses are contained on the following pages.

The BLANK WHEN ZERO clause is used to enable the blanking of an item when that item's value is zero. The clause is optional.

General Format:

BLANK WHEN ZERO

Syntax Rules:

1. The BLANK WHEN ZERO clause may be used only for an elementary item whose PICTURE is specified as numeric or numeric edited.
2. This clause cannot be used for variable-length items.

General Rules:

1. When this clause is used, the item contains only spaces when the value of the item is zero.
2. When used for an item whose PICTURE is numeric, the category of the item is considered to be numeric edited.

BLOCK CONTAINS

The BLOCK CONTAINS clause in an FD entry is used to specify the size of a physical record. This clause is optional and may be omitted when the file has the standard physical record size of 320 computer words.

General Format:

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

Syntax Rule:

1. Integer-1 (when used) and integer-2 must be unsigned nonzero integers.

General Rules:

1. For mass storage files, the size of a physical record may be given in terms of records only if one of the following conditions does not exist. The CHARACTERS option must be specified if any of the following conditions exist.

   a. Logical records extend across physical records.

   b. The physical record contains padding (area not contained in a logical record).

   c. Logical records are grouped so that an inaccurate physical record size would be implied.

2. Whether the block size is given in terms of characters or recor s, each logical record will begin in a new computer word.

3. When the CHARACTERS option is used, the physical record size is specified in terms of the number of standard characters contained in the physical record, regardless of the types of characters used to represent the items within the physical record.

4. If only integer-2 is shown, it represents the exact size of the physical record. If both integer-1 and integer-2 are given, they refer to the minimum and maximum size of the physical record, respectively. In that case, integer-1 is for documentation purposes only.

5. The use of the word CHARACTERS in the clause is optional. Whenever the keyword RECORDS is not specifically written in the clause, integer-1 (if used) and integer-2 represent the number of characters in the block. The word RECORDS cannot be used for a file for which REPORT(S) is also specified.

6. When the RECORDS phrase is used with variable-length records, the block size is equal to the maximum record size (in computer words) multiplied by the number of records plus one.

7. For mass storage files in the RANDOM ACCESS mode, the physical record size associated with the CHARACTERS option is considered identical with the logical record size. Depending on the mass storage device normally intended for the file, the physical record size may be adjusted for efficiency by using this clause with the CHARACTERS option. Integer-2 may range from 384 to 24,570 characters (which results in record sizes in the range 64 to 4095 words). A size that is not modulo 384 characters will result in wasted space.

The CODE clause of the RD entry is used to affix a unique character to each report group generated in the report. The unique character identifies each print line as belonging to a specific report.

General Format:

CODE mnemonic-name-1

General Rules:

1. CODE mnemonic-name-1 indicates a unique character(s) which is automatically affixed to and identifies each line of the report. More than one report may then be produced simultaneously onto one output device for later individual report selection.

2. The mnemonic-name must be identified with a unique character in the SPECIAL-NAMES paragraph of the Environment Division. Unique character assignment must be assured when the program is to produce more than one report.

3. As shown in the complete entry skeleton for the report description, the CODE clause, when used, must immediately follow the report-name.

The COLUMN NUMBER clause in a report group description entry is used to indicate the absolute column number on the printed page of the leftmost character of the elementary item (the first print position of the item on the line).

General Format:

COLUMN NUMBER IS integer-1

Syntax Rules:

1. The COLUMN NUMBER clause is used only at the elementary level within a report group.

2. Integer-1 must be an unsigned and nonzero integer. The first position of the print line is considered to be COLUMN NUMBER 1.

3. For any given LINE NUMBER specification within a report group, COLUMN NUMBER entries must be indicated from left to right.

General Rule:

1. The COLUMN NUMBER clause indicates that the associated elementary item is presented in the output report group. If COLUMN NUMBER is not specified for an elementary item, that elementary item is included in the description of the report group for control purposes but is not presented when the report group is produced at object program execution.

The CONTROL(S) clause in the report description (RD) entry is used to indicate the identifiers which specify the control hierarchy (the control breaks) for this report.

General Format:

```
{ CONTROL IS  }   { FINAL                                          }
{ CONTROLS ARE}   { identifier-1 [, identifier-2] ...              }
                  { FINAL, identifier-1 [ , identifier-2] ...      }
```

Syntax Rules:

1. The identifiers specify the control hierarchy for this report and are listed in order from major to minor. FINAL is the highest control, identifier-1 is the major control, identifier-2 is the intermediate control, etc. The last identifier specified is the minor control.

2. The identifiers used in the CONTROL clause must be defined in the File Section or Working-Storage Section of the Data Division.

General Rules:

1. FINAL is a reserved word that indicates the highest control for this report. A TYPE CONTROL HEADING FINAL report group is only produced once, at the beginning of the report, when the first GENERATE statement is executed. Similarly, a TYPE CONTROL FOOTING FINAL report group is produced only once, at the end of a report, when the TERMINATE statement is executed.

2. The CONTROL clause is required when CONTROL HEADING or CONTROL FOOTING report groups are specified.

3. The identifiers specified in the CONTROL clause are the only identifiers referred to by the RESET and TYPE clauses of a report group description entry for this report. An identifier used in the CONTROL clause cannot be referred to by more than one TYPE CONTROL HEADING report group and one TYPE CONTROL FOOTING report group.

The COPY clause is used in association with a file or data-name in FD, SD, RD, record description, and report group description entries. The COPY clause is used to direct the compiler to duplicate text from the source program or a library into the source program.

Format 1:

```
COPY   library-name

[ REPLACING   word-1   BY  { word-2      }
                          { identifier-1 }
                          { literal-1    }

    [ , word-3   BY   { word-4      }      ]   ... ] .
                      { identifier-2 }
                      { literal-2    }
```

Format 2:

```
{ level indicator }
{ level-number    }  data-name-1  COPY  data-name-2  [ FROM LIBRARY ].
```

Syntax Rules:

1. In Format 1, when the COPY clause is specified, the library-name is required. The library-name must be identical to the name associated with the desired text on the library.

2. In Format 1, a word is any COBOL word and may be one of the following:

   - Condition-name
   - Data-name
   - File-name
   - Mnemonic-name

3. In Format 1, the COPY clause may be specified only at the 01 level in data description entries and report group description entries.

General Rules:

1. Format 1 of the COPY clause represents the American National Standard COPY function. Format 2 represents the HIS COPY function.

2. For a detailed description of the COPY clause, see Section VIII, the COBOL Library, and Section XIV of the COBOL User's Guide.

data-name/FILLER

The data-name clause is used to specify the name of the data being described. The FILLER clause is used to specify an elementary item of the logical record that cannot be referred to directly.

General Format:

level-number { data-name / FILLER }

Syntax Rules:

1. In the File Section and Working-Storage Section, a data-name or the keyword FILLER must be the first word following the level-number in each data description entry.

2. In the Report Section, a data-name need not appear in a data description entry and the word FILLER cannot be used.

General Rules:

1. The keyword FILLER can be used to name an elementary item in a record, but that item then cannot be referred to directly under any circumstances.

2. A data-name must be supplied in the Report Section when:

   a. The data-name represents a report group to be referred to by a GENERATE or a USE statement in the Procedure Division.

   b. Reference is to be made to the SUM counter in the Procedure Division or in the Report Section.

   c. The SELECTED option is included with the SOURCE clause at a higher level to indicate that at this lower level the SOURCE data-names which are to be used are elementary items.

DATA RECORDS

The DATA RECORDS clause is used to provide documentation for the names of data records within their associated files (SD or FD entries).

General Format:

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

Syntax Rule:

1. Data-name-1 and data-name-2 are the names of data records. They must have level-number 01 data descriptions (with the same names) associated with them.

General Rules:

1. The presence of more than one data-name indicates that the file contains more than one type of data record. If record sizes (in words) are not equal, the file is assigned the variable-length record format and its recording mode must be binary. The records of the file need not have the same description. The order in which they occur as 01 entries is not significant except for sort files. For a sort file with more than one size data record description, the first record description entry after the SD entry is assumed to be the dominant type; its size is considered to be the most common in the sort file. Sort optimization is based on this assumption. Therefore, a careful choice in ordering record description entries for a sort file enhances object program efficiency.

2. Conceptually, all data records in a file share the same area. This concept is not altered by the presence of more than one type of data record in a file.

The GROUP INDICATE clause in a report group description entry is used to indicate that the elementary item with which it is associated is to be produced only on the first occurrence of the item after any CONTROL or PAGE break.

General Format:

GROUP INDICATE

General Rules:

1. The GROUP INDICATE clause must be specified only at the elementary item level within a TYPE DETAIL report group.

2. When an elementary item within a TYPE DETAIL report group is specified as GROUP INDICATE, it is presented in the first DETAIL report group that contains it after a CONTROL break and it is presented in the first DETAIL report group containing it on a new page even though a CONTROL break did not occur.

The JUSTIFIED clause in a data description entry is used to specify nonstandard positioning of data within a receiving data item.

General Format:

$$\left\{\begin{array}{l}\underline{\text{JUSTIFIED}}\\ \underline{\text{JUST}}\end{array}\right\} \text{RIGHT}$$

Syntax Rules:

1. The JUSTIFIED clause can be specified only at the elementary item level.
2. JUST is an abbreviation for JUSTIFIED.

General Rules:

1. The JUSTIFIED clause cannot be specified for an item that has any of the following properties:
   a. Class numeric or numeric edited.
   b. USAGE other than DISPLAY.
   c. Actual or assumed decimal point.
2. If the receiving data item is alphanumeric (other than a numeric edited data item) or alphabetic and the JUSTIFIED clause is not specified, the sending data item is moved to the receiving character positions and aligned at the leftmost character position in the data item with space-fill or truncation to the right.
3. When the receiving data item is described with the JUSTIFIED clause and the sending data item is larger than the receiving data item, the leftmost characters are truncated. When the receiving data item is described with the JUSTIFIED clause and the sending data item is smaller than the receiving data item, the data is aligned at the rightmost character position in the data item with space-fill on the left.

LABEL RECORDS

The LABEL RECORD(S) clause in the FD entry is used to state whether or not label records are present and to identify them if they are present.

General Format:

```
                  ┌ RECORD IS  ┐   ┌ STANDARD                                          ┐
* LABEL           │            │   │ OMITTED                                           │
                  └ RECORDS ARE┘   └ label-name-1 [ , label-name-2 ] ...               ┘
```

Syntax Rules:

1. The LABEL RECORD(S) clause is required in every file description entry.

2. If one or more label-names are specified, only the four fixed label-names listed in general rule 5 may be used.

General Rules:

1. LABEL RECORD(S) STANDARD indicates that logical labels that conform to the Series 60/6000 label format specifications are considered to exist for the file even though they may not be recorded on some of the physical devices to which the file may be assigned. (For example, label records associated with mass storage files are not physically recorded on the external storage device. In this case, USE procedures associated with the file will act upon dummy label space. If identifying information is to be placed on the external storage device, the information must be written as if it were a data record.) The STANDARD option must be used whenever system standard format has been explicitly or implicitly specified for the file.

2. LABEL RECORD(S) OMITTED indicates that no logical labels are considered to exist for a file. The OMITTED option must not be used if system standard format has been explicitly or implicitly specified for the file.

3. All Procedure Division references to label-name-1, or to any items subordinate to label-name-1, must appear within USE procedures.

4. On a multiple file tape, either all files must be labeled or no files can be labeled.

LABEL RECORDS

5. A label-name must be one of the four fixed label-names associated with the Series 60/6000 standard label format specifications. These label-names and their associated Series 60/6000 standard label records are described below:

   a. BEGINNING-FILE-LABEL: Appears only once for a file, preceding the first data record of the file, and contains information about the file.

   b. BEGINNING-TAPE-LABEL: Appears at the physical beginning of each reel, with the exception of the first reel on the file, preceding all other information, and contains information about the reel.

   c. ENDING-FILE-LABEL: Appears only once for a file, following the last data record on the last reel of a file, and contains information about the file.

   d. ENDING-TAPE-LABEL: Appears at the physical end of a reel, with the exception of the last reel of the file, following the last data record, and contains information about the reel.

   Since these fixed label-names are associated with the Series 60/6000 standard label descriptions, their formats are inferred by the compiler and it is not necessary to define them within the file description. If they are explicitly described in the source program, the specified record format will be used.

6. Conceptually, all label records within the file share the same label buffer area in memory. If more than one label-name is specified, it is an implicit redefinition of the same label area.

7. Refer to the description of the LABEL RECORD(S) clause in Section III of the COBOL User's Guide for additional information.

The level-number entries show the hierarchy of data within a logical record or report group. Level-number entries are also used to identify entries for condition-names, working-storage items, noncontiguous data items, and the RENAMES clause.

The concepts of hierarchy of data and the use of level-numbers are described in Section III.

General Format:

level-number

Syntax Rules:

1. A level-number is required as the first element in each data description entry.

2. Data description entries that are subordinate to FD or SD entries may use level-numbers with values in the range 01 (or 1) through 49, or level-number 66, or level-number 88.

3. Data description entries that are subordinate to an RD entry may only use level-numbers whose values are in the range 01 through 49.

4. Multiple level 01 entries that are subordinate to a level indicator (FD, SD, or RD), represent implicit redefinitions of the same area. In other words, the first occurrence defines an area, and all subsequent occurrences redefine the same area.

General Rules:

1. A level 01 entry must be used to identify the first entry in each record description and in each report group description.

2. When no real concept of level exists, the special level-numbers 66, 77, and 88 are assigned as follows:

   a. Level-number 66 is used in a data description entry to identify a RENAMES entry. Format 4 of the data description skeleton must be used.

   b. Level-number 77 entries may be used in the Working-Storage Section to identify noncontiguous data items. Format 3 of the data description skeleton must be used.

   c. Level-number 88 entries are used to define condition-names associated with conditional variables and may be used in the File Section and the Working-Storage Section. Format 5 of the data description skeleton must be used.

The LINE NUMBER clause is used to indicate the line number of this entry. The line number may be absolute or relative and may refer either to the page or to the previous entry.

General Format:

```
LINE NUMBER IS { integer-1          }
               { PLUS integer-2     }
               { NEXT PAGE          }
```

Syntax Rules:

1. Integer-1 and integer-2 must be unsigned nonzero integers. Integer-1 must be within the range specified in the PAGE LIMITS clause in the associated RD entry.

2. The LINE NUMBER clause must be given for each report line of a report group. For the first line of a report group, the LINE NUMBER clause must be specified at the report group level, or at a group level subordinate to the report group level but prior to the first elementary item in the line, or as a part of the entry which describes the first elementary item in the line.

   For report lines other than the first line in a report group, the LINE NUMBER clause must be specified at a group level subordinate to the report group level but prior to the first elementary item in the line, or as a part of the entry which describes the first elementary item in the line.

General Rules:

1. Integer-1 indicates an absolute line number. The LINE-COUNTER is set to the value of integer-1 for printing the item in this entry (and following entries in the report group) until a different value for the LINE-COUNTER is specified in another LINE NUMBER clause.

2. Integer-2 indicates a relative line number. This number increments the LINE-COUNTER for printing the item in this entry (and following entries in the report group) until a different value for the LINE-COUNTER is specified in another LINE NUMBER clause.

3. If the LINE NUMBER clause is specified at the report group level, entries for the first report line within the report group are presented on the specified line number. If the LINE NUMBER clause is specified for an entry on other than the report group level, sequential entries following that entry within the report group that have the same level-number are presented on the same line. A line number at a subordinate level cannot contradict a line number at a group level.

4. Within a report group description entry, an absolute line number cannot be preceded by a relative line number and absolute line numbers must be given in ascending order.

5. The NEXT PAGE phrase is used to indicate an automatic skip to the next page before the first line of the current report group is presented. Appropriate TYPE PAGE/OVERFLOW FOOTINGS and TYPE PAGE/OVERFLOW HEADINGS will be produced as specified.

The NEXT GROUP clause in a report group description entry is used to indicate the spacing that is to follow the last line of the report group.

General Format:

```
NEXT GROUP IS { integer-1          }
              { PLUS integer-2     }
              { NEXT PAGE          }
```

Syntax Rule:

1. Integer-1 and integer-2 must be unsigned nonzero integers. When used, integer-1 cannot exceed the maximum number of lines specified per report page in the PAGE LIMITS clause of the associated RD entry.

General Rules:

1. Integer-1 indicates an absolute line number that sets the LINE-COUNTER to this value after producing the last line of the current report group.

2. Integer-2 indicates a relative line number that increments the LINE-COUNTER by the value of integer-2. Integer-2, therefore, represents the number of lines to be skipped following the last line of the current report group. Further spacing is specified by the LINE NUMBER clause of the next report group produced.

3. The NEXT PAGE phrase, when used, indicates an automatic skip to the next page following the generation of the last line of the current report group. Appropriate PAGE/OVERFLOW FOOTINGS and PAGE/OVERFLOW HEADINGS will be produced as specified.

4. The NEXT GROUP clause may appear only in a level 01 entry that defines the report group. When specified for a CONTROL FOOTING/HEADING report group, the NEXT GROUP clause results in automatic line spacing only when a control break occurs on the level for which that control is specified.

The OCCURS clause in a data description entry is used to define tables of repeated items. Use of this clause eliminates the need for separate entries for repeated data and supplies information needed for the application of subscripts or indexes.

Format 1:

```
OCCURS integer-2 TIMES

[ { ASCENDING  } KEY IS data-name-2 [ , data-name-3 ] ... ] ...
  { DESCENDING }

[ INDEXED BY index-name-1 [ , index-name-2 ] ... ]
```

Format 2:

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

[ { ASCENDING  } KEY IS data-name-2 [ , data-name-3 ] ... ] ...
  { DESCENDING }

[ INDEXED BY index-name-1 [ , index-name-2 ] ... ]
```

Syntax Rules:

1. Integer-1 and integer-2 must be positive integers. Where both are used (in Format 2), the value of integer-1 must be less than the value of integer-2. The value of integer-1 may be zero but integer-2 must not be zero.

2. Data-name-1 (in the DEPENDING phrase) must describe a positive integer with USAGE COMP-1.

3. Data-names used in the OCCURS clause may be qualified.

4. Data-name-2 must either be the name of the entry containing the OCCURS clause or the name of an entry subordinate to the entry containing the OCCURS clause.

5. When used, data-name-3, etc., must be the name of an entry subordinate to the group item that is the subject of this data description entry.

6. The INDEXED BY phrase is required when the subject of this data description entry (or an entry subordinate to this entry if it is a group item) is to be referred to by indexing. The index-names identified in the OCCURS clause are not defined elsewhere since the allocation and format of these index-names is hardware dependent. Not being data, these index-names cannot be associated with any data hierarchy.

7. The DEPENDING ON phrase is required only when the end of the occurrences of the item cannot otherwise be determined.

8. The OCCURS clause may not be specified in a data description entry that has a level-number 01, 66, 77, 88, or in an entry that describes an item whose size is variable. An item is considered to be variable in size if its data description, or the description of any item subordinate to it, if it is a group item, uses Format 2 of the OCCURS clause.

9. Data-name-1 (in the DEPENDING phrase) must be an entry in the same record as the current data description entry, and the level-number entry for data-name-1 must occur prior to the level-number entry containing the OCCURS clause in which data-name-1 is used.

10. Any entry that contains, or has a subordinate entry that contains, Format 2 of the OCCURS clause, cannot be the object of a REDEFINES clause.

11. If data-name-2 in the KEY phrase is not the subject of this entry, then:

    a. All of the items identified by the data-names used in the KEY phrase must be contained in the group item that is the subject of this entry.

    b. The items identified by the data-names in the KEY phrase may not be described by an entry that contains an OCCURS clause or be subordinate to an entry containing an OCCURS clause.

12. Index-name-1, index-name-2, ..., must be unique words within the program.

General Rules:

1. The OCCURS clause is used to define tables and other homogeneous sets of repeated data. The data-name that is the subject of this data description entry must either be subscripted or indexed whenever it is referred to in a Procedure Division statement other than the SEARCH statement. In addition, if the subject of this data description entry is the name of a group item, all of the data-names belonging to the group must be subscripted or indexed whenever they are used as operands.

2. All data description entry clauses except the OCCURS clause itself associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described.

3. In Format 1, the value of integer-2 represents the exact number of occurrences. In Format 2, the value of integer-2 represents the maximum number of occurrences and the value of integer-1 represents the minimum number of occurrences. Thus, Format 2 specifies a variable number of occurrences of the item but does not imply that the length of the item is variable.

4. The value of data-name-1 is the count of the number of occurrences of the subject and this value must not exceed integer-2. Reducing the value of data-name-1 makes the contents of those occurrences of the data items, whose occurrence numbers are in excess of the value of the data item referenced by data-name-1, unpredictable.

5. When a referenced group item has a subordinate entry that specifies Format 2 of the OCCURS clause, only that part of the table area that is specified by the value of data-name-1 will be used in the operation.

6. The results of OCCURS...DEPENDING generally differ from one computer to another. In Series 60/6000 COBOL, the results are as follows:

   a. The DEPENDING phrase may not be specified in the record descriptions of a sort file or a merge file.

   b. In the File Section, OCCURS...DEPENDING results in suppressing table residue on the peripheral device, as described below, when the following criteria are met:

      - Data-name-1 is described as COMP-1.
      - Data-name-1 is subordinate to the same record description entry.
      - Data-name-1 precedes the data description entry containing the OCCURS...DEPENDING clause.

      When compression is to take place, the compiler will automatically generate a process area for the file, whether or not the APPLY PROCESS AREA phrase is specified in the I-O-CONTROL paragraph.

   c. When a WRITE statement references a record which may be compressed, the object program examines the output record for opportunities for residue suppression. Such opportunities are rejected unless two or more machine words can be suppressed. In the latter case, suppression proceeds on a machine word basis; the whole-word portion of the residue of each table is replaced by a single control word. Each variable-length table in the record presents an opportunity for residue suppression. Actual suppression takes place in an implicit move from the process area to the output buffer.

d. When a READ statement references a file containing records that may be compressed, the record is implicitly moved from the input buffer to the process area and expanded to the format it had in memory prior to residue suppression.

e. When OCCURS...DEPENDING is used with any record of a file that may be compressed, the data format is affected in several ways.

- The variable-length record (VLR) format is automatically applied.
- In addition to any residue suppression control words needed, each record on the peripheral device begins with a control word required for reconstruction.
- The recording mode must be binary (explicitly or implicitly).

7. The KEY phrase is used to indicate that the repeated data is arranged in ascending or descending order according to the values contained in data-name-2, data-name-3, etc. The data-names are listed in their descending order of significance, from most significant to least significant.

The PAGE LIMIT(S) clause in a report description entry is used to indicate the specific line control to be maintained within the logical presentation of a page.

General Format:

```
PAGE { LIMIT IS  } integer-1 { LINE  }
     { LIMITS ARE}           { LINES }

[, HEADING integer-2 ] [, FIRST DETAIL integer-3]

[, LAST DETAIL integer-4] [, FOOTING integer-5 ]
```

Syntax Rules:

1. Integer-1 through integer-5 must be unsigned nonzero integers.
2. Integer-2 through integer-5 each must either be less than or equal to integer-1.
3. Only one PAGE-LIMIT clause may be specified in each report description entry.
4. A PAGE LIMIT(S) clause need not be included in the RD entry when an association between report groups and the physical format of a page is not required.

General Rules:

1. The fixed data-names PAGE-COUNTER and LINE-COUNTER are automatically generated by the Report Writer when the PAGE LIMIT(S) clause is included in the RD entry.
2. The PAGE LIMIT integer-1 LINES clause is required to specify the depth of the report page; the depth of the report page may or may not be equal to the physical perforated continuous form often associated with the page length in a report.
3. LINE-COUNTER must be capable of containing the value specified by integer-1.
4. If absolute line spacing is indicated for all the report group(s), none of the integer-2 through integer-5 controls need to be specified.

5. If relative spacing is indicated for individual TYPE DETAIL report group description entries, some or all of the above limits must be defined, dependent on the type of report groups within the report, so that the Report Writer can maintain control of page format.

   - HEADING integer-2: the first line number of the first heading print group. No print group will start preceding integer-2.
   - FIRST DETAIL integer-3: the first line number of the first normal print group, that is, body; no DETAIL or CONTROL print group will start before integer-3.
   - LAST DETAIL integer-4: the last line number of the last normal print group, that is, body; no DETAIL or CONTROL HEADING print group will extend beyond integer-4.
   - FOOTING integer-5: the last line number of the last CONTROL FOOTING print group; no CONTROL FOOTING print group will start before integer-3 nor extend beyond integer-5. TYPE PAGE FOOTING or TYPE OVERFLOW FOOTING print groups will follow integer-5.

6. When relative line numbers are specified for report groups, PAGE LIMITS integer-1 is specified and some or all of the HEADING integer-2, FIRST DETAIL integer-3, LAST DETAIL integer-4, or FOOTING integer-5 phrases are omitted, the following implicit control is assumed for the omitted specifications:

   a. If HEADING integer-2 is omitted, integer-2 is considered to be equivalent to the value one (1), that is, line number one.

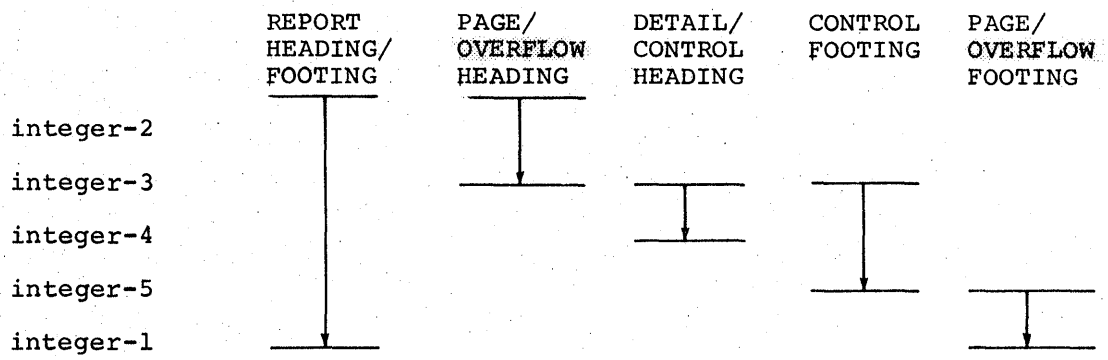
   b. If FIRST DETAIL integer-3 is omitted, integer-3 is considered to be equivalent to the value of integer-2.

   c. If LAST DETAIL integer-4 is omitted, integer-4 is considered to be equivalent to the value of integer-5.

   d. If FOOTING integer-5 is omitted, integer-5 is considered to be equivalent to the value of integer-4.

   e. If both LAST DETAIL integer-4 and FOOTING integer-5 are omitted, integer-4 and integer-5 are both considered to be equivalent to the value of integer-1.

7. The following chart represents page format report group control when the PAGE LIMIT clause is specified.



8. Absolute LINE NUMBER or absolute NEXT GROUP spacing (see report group description entry) must be consistent with controls specified in the PAGE LIMIT clause.

PAGE-COUNTER Rules:

1. PAGE-COUNTER is a fixed data-name used to reference a counter generated by the Report Writer to be used as a SOURCE data item in order to automatically present consecutive page numbers.

2. One PAGE-COUNTER is supplied for each report described in the Report Section.

3. If more than one PAGE-COUNTER is given as a SOURCE data item within a given report, the number of numeric characters indicated by the PICTURE clauses must be identical. The size must indicate sufficient numeric character positions to prevent overflow.

4. If more than one report description entry exists in the Report Section, PAGE-COUNTER must be qualified by the report-name. PAGE-COUNTER may be referred to in Data Division clauses and in Procedure Division statements.

5. PAGE-COUNTER is initially set to one by the Report Writer; if a starting value for PAGE-COUNTER other than one is desired, the user may change the contents of the PAGE-COUNTER by a Procedure Division statement after an INITIATE statement has been executed.

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

LINE-COUNTER Rules:

1. LINE-COUNTER is a fixed data-name used to reference a counter utilized by the Report Writer to determine when a PAGE/OVERFLOW HEADING and/or a PAGE/OVERFLOW FOOTING report group is to be presented. If a PAGE LIMIT(S) clause is written in the report description entry, a LINE-COUNTER is supplied for that report.

2. If more than one report description entry exists in the Report Section, LINE-COUNTER must be qualified by the report-name. LINE-COUNTER may be referred to in Data Division clauses and in Procedure Division statements.

3. Changing the LINE-COUNTER by Procedure Division statements may cause page format control in the Report Writer to become unpredictable.

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

5. LINE-COUNTER is initially set to zero by the Report Writer; likewise, LINE-COUNTER is automatically reset to zero when the PAGE LIMIT integer-1 LINES entry is exceeded during execution.

6. When a relative LINE NUMBER indication or relative NEXT GROUP indication exceeds the LAST DETAIL PAGE LIMIT specification during object program execution, a page break occurs and LINE-COUNTER is reset to zero. No additional setting based on the relative LINE NUMBER indication or NEXT GROUP indication that forced the page break takes place.

7. If an absolute LINE NUMBER indication or an absolute NEXT GROUP indication is equal to, or less than, the contents of the LINE-COUNTER during object program execution, the LINE-COUNTER is set to the absolute LINE NUMBER indication or the absolute NEXT GROUP indication following the implicit generation of any specified report groups.

8. The value of the LINE-COUNTER during any Procedure Division test statement represents the number of the last line used by the printing generated by the previous report group, or represents the number of the last line skipped by a previous NEXT GROUP specification.

The PICTURE clause is used to describe the general characteristics and editing requirements of an elementary item.

General Format:

$$\left\{\begin{matrix}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{matrix}\right\} \text{ IS character-string}$$

Syntax Rules:

1. A PICTURE clause may be specified only at the elementary item level.
2. A character-string consists of certain allowable combinations of characters in the COBOL character set used as symbols. The allowable combinations determine the category of the elementary item.
3. The maximum number of symbols allowed in the character-string is 30.
4. The PICTURE clause must be specified for every elementary data item except an index data item, in which case use of this clause is prohibited.
5. PIC is an abbreviation for PICTURE.
6. The asterisk, when used as the zero suppression symbol, and the BLANK WHEN ZERO clause may not appear in the same entry.

General Rules:

1. Five categories of data may be described with a PICTURE clause; alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited.
   - To define an item as alphabetic:
     a. Its PICTURE character-string may only contain the symbol 'A'; and
     b. Its contents when represented in standard data format must be any combination of the twenty-six (26) letters of the Roman alphabet and the space from the COBOL character set.
   - To define an item as numeric:
     a. Its PICTURE character-string can only contain the symbols '9', 'P', 'S', and 'V'; and
     b. Its contents when represented in standard data format must be a combination of the Arabic numerals '0', '1', '2', '3', '4', '5', '6', '7', '8', and '9', and the item may include an operational sign.

- To define an item as alphanumeric:

  a. Its PICTURE character-string is restricted to certain combinations of the symbols 'A', 'J', 'K', 'X', '9', and the item is treated as if the character-string contained all 'X's. A PICTURE character-string which contains all 'A's or all '9's, with or without the symbols 'J' or 'K', does not define an alphanumeric item; and

  b. Its contents when represented in standard data format are allowable characters in the computer's character set.

- To define an item as alphanumeric edited:

  a. Its PICTURE character-string is restricted to certain combinations of the following symbols: 'A', 'X', '9', 'B', and '0'; and

     (1) The character-string must contain at least one 'X' and at least one 'B' or '0' (zero); or

     (2) The character-string must contain at least one '0' (zero) and at least one 'A'; and

  b. Its contents when represented in standard data format are allowable characters in the computer's character set.

- To define an item as numeric edited:

  a. Its PICTURE character-string is restricted to certain combinations of the symbols 'B', 'P', 'V', 'Z', '0', '9', ',', '.', '*', '+', '-', 'CR', 'DB', and the currency symbol. The maximum number of digit positions that may be represented in the character-string is 18; and

  b. The contents of the character positions of those symbols allowed to represent a digit in standard data format must be one of the numerals.

2. The size of an elementary item, where size means the number of character positions occupied by the elementary item in standard data format, is determined by the number of allowable symbols that represent character positions. An unsigned nonzero integer which is enclosed in parentheses following the symbols 'A', ',', 'X', '9', 'P', 'Z', '*', 'B', '0', '+', '-', or the currency symbol indicates the number of consecutive occurrences of the symbol. The following symbols may appear only once in a given PICTURE: 'S', 'V', '.', 'CR', and 'DB'.

3. The functions of the symbols used to describe an elementary item are explained as follows:

   A Each 'A' in the character-string represents a character position which can contain only a letter of the alphabet or a space.

   B Each 'B' in the character-string represents a character position into which the space character will be inserted.

J,K Series 60/6000 COBOL accepts J and K as equivalent to X. However, the initial value of a working-storage item is 'spaces' if PICTURE contains a J and no VALUE clause is given. This feature simplifies conversion from certain other computers, and is not meant as a general substitute for VALUE SPACES.

P The 'P' indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character 'P' is not counted in the size of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric edited items or numeric items which appear as operands in arithmetic statements. (P cannot be used with COMPUTATIONAL or COMPUTATIONAL-n items.) The scaling position character 'P' can appear only to the left or right as a continuous string of 'P's within a PICTURE description. Since the scaling position character 'P' implies an assumed decimal point (to the left of 'P's if 'P's are leftmost PICTURE characters and to the right of 'P's if 'P's are rightmost PICTURE characters), the assumed decimal point symbol 'V' is redundant as either the leftmost or rightmost character within such a PICTURE description.

S The letter 'S' is used in a character-string to indicate the presence of an operational sign and must be written as the leftmost character in the PICTURE character-string. The 'S' is not counted in determining the size of the elementary item unless the PICTURE clause is accompanied by a USAGE COMP-4 clause. In this case, the 'S' is counted as a separate character in the size of the elementary item. Refer to the USAGE clause for additional information.

V The 'V' is used in a character-string to indicate the location of the assumed decimal point and may only appear once in a character-string. The 'V' does not represent a character position and therefore is not counted in the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string, the 'V' is redundant.

X Each 'X' in the character-string is used to represent a character position which contains any allowable character from the computer's character set. If a PICTURE character-string contains combinations of X, A, and 9's, then the PICTURE is treated as if it consisted entirely of X's.

Z Each 'Z' in a character-string may only be used to represent the leftmost leading numeric character positions which will be replaced by a space character when the contents of that character position are zero. Each 'Z' is counted in the size of the item.

9 Each '9' in the character-string represents a character position which contains a numeral and is counted in the size of the item.

0 Each '0' (zero) in the character-string represents a character position into which the numeral zero will be inserted. The '0' is counted in the size of the item.

, Each ',' (comma) in the character-string represents a character position into which the character ',' will be inserted. This character position is counted in the size of the item.

. When the character '.' (period) appears in the character-string, it is an editing symbol which represents the decimal point for alignment purposes and, in addition, represents a character position into which the character '.' will be inserted. The character '.' is counted in the size of the item. The symbols 'V' and '.' are mutually exclusive. For a given program, the functions of the period and comma are exchanged if the phrase DECIMAL-POINT IS COMMA is specified in the SPECIAL-NAMES paragraph. In this exchange, the rules for the period apply to the comma and the rules for the comma apply to the period wherever they appear in a PICTURE clause. (If insertion character '.' is the last symbol in the PICTURE character-string, it must be immediately followed by a semicolon or the period to end the data description entry.)

+,-,CR,DB These symbols are used as editing sign control symbols. When used, they represent the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in any one character-string and each character used in the symbol is counted in determining the size of the data item.

* Each '*' (asterisk) in the character-string represents a leading numeric character position into which an asterisk will be placed when the contents of that position are zero. Each '*' is counted in the size of the item.

$ The '$' (currency symbol) in the character-string represents a character position into which a currency symbol is to be placed. The currency symbol in a character-string is represented by either the $ or by the single character specified in the CURRENCY SIGN phrase in the SPECIAL-NAMES paragraph. The currency symbol is counted in the size of the item.

Editing Rules:

1. Two general methods are used to perform editing in the PICTURE clause, either by insertion or by suppression and replacement. The four types of insertion editing are:

   a. Simple insertion.

   b. Special insertion.

   c. Fixed insertion.

   d. Floating insertion.

The two types of suppression and replacement editing are:

a. Zero suppression and replacement with spaces.

b. Zero suppression and replacement with asterisks.

2. The type of editing which may be performed upon an item depends on the category to which the item belongs. The following list indicates which type of editing may be performed upon a given category:

Category	Type of Editing
Alphabetic	None
Numeric	None
Alphanumeric	None
Alphanumeric Edited	Simple insertion, '0' and 'B'
Numeric Edited	All, subject to the restrictions of Rule 3 below
Any Variable-Length Item	None

3. Floating insertion editing and editing by zero suppression and replacement are mutually exclusive in a PICTURE clause. Only one type of replacement may be used with zero suppression in a PICTURE clause.

4. Simple Insertion Editing.

The ',' (comma), 'B' (space), and '0' (zero) are used as the insertion characters. The insertion characters are counted in the size of the item and represent the position in the item into which the character will be inserted.

5. Special Insertion Editing.

The '.' (period) is used as the insertion character and also represents the decimal point for alignment purposes. The insertion character used for the actual decimal point is counted in the size of the item. The use of the assumed decimal point, represented by the symbol 'V', and the actual decimal point, represented by the insertion character, in the same PICTURE character-string is not allowed. If the insertion character is the last symbol in the character-string and additional clauses follow the character-string, then the character-string must be immediately followed by the semicolon punctuation character, followed by a space. If the PICTURE clause is the last clause of that Data Division entry, and the insertion character is the last symbol in the character-string, the insertion character must be immediately followed by a period punctuation character, followed by a space. This results in two consecutive periods appearing in the data description entry. The result of special insertion editing is the appearance of the insertion character in the item in the same position as shown in the character-string.

6. Fixed Insertion Editing.

The currency symbol and the editing sign control symbols '+', '-', 'CR', 'DB' are the insertion characters. Only one currency symbol and only one of the editing sign control symbols can be used in a given

PICTURE character-string. When the symbols 'CR' or 'DB' are used, they represent two character positions in determining the size of the item and they must represent the rightmost character positions that are counted in the size of the item. The symbol '+' or '-', when used, must be the leftmost or rightmost character position to be counted in the size of the item. The currency symbol must be the leftmost character position to be counted in the size of the item except that it may be preceded by either a '+' or a '-' symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupied in the PICTURE character-string. Editing sign control symbols produce the following results depending upon the value of the data item.

Editing Symbol in PICTURE Character-String	RESULT	
	Data Item Positive or Zero	Data Item Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

7. Floating Insertion Editing.

The currency symbol and editing sign control symbols '+' or '-' are the insertion characters and they are mutually exclusive as floating insertion characters in a given PICTURE character-string.

Floating insertion editing is indicated in a PICTURE character-string by using a string of at least two of the allowable insertion characters to represent the leftmost numeric character positions into which the insertion characters can be floated. Any of the simple insertion characters embedded in the string of floating insertion characters or to the immediate right of this string are part of the floating string.

In a PICTURE character-string, there are only two ways of representing floating insertion editing. One is to represent any or all of the leading numeric character positions on the left of the decimal point by the insertion character. The other is to represent all of the numeric character positions in the PICTURE character-string by the insertion character.

If the insertion characters are only to the left of the decimal point in the PICTURE character-string, the result is that a single insertion character will be placed into either the character position represented by the rightmost occurrence of the insertion character in the PICTURE character-string or the character position immediately preceding the first nonzero digit in the data represented by the insertion symbol string, whichever is farther to the left in the PICTURE character-string. The character positions preceding the insertion character are filled with spaces.

If all numeric character positions in the PICTURE character-string are represented by the insertion character, the result depends upon the value of the data. If the value is zero, the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion character is only to the left of the decimal point in the PICTURE character-string.

To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item must be the number of characters in the sending data item, plus the number of fixed insertion characters being edited into the receiving data item, plus one for the floating insertion character.

8. Zero Suppression Editing.

The suppression of leading zeros in numeric character positions is indicated by the use of the alphabetic character 'Z' or the character '*' (asterisk) as suppression symbols in a PICTURE character-string. These symbols are mutually exclusive in a given PICTURE character-string. Each suppression symbol is counted in determining the size of the item. If 'Z' is used, the replacement character will be the space; if the asterisk is used, the replacement character will be '*'.

Zero suppression and replacement is indicated in a PICTURE character-string by using a string of one or more of the allowable symbols to represent leading numeric character positions which are to be replaced when the associated character position in the data contains a zero. Any of the simple insertion characters embedded in the string of symbols or to the immediate right of this string are part of the string.

In a PICTURE character-string, there are only two ways of representing zero suppression. One is to represent any or all of the leading numeric character positions to the left of the decimal point by suppression symbols. The other is to represent all of the numeric character positions in the PICTURE character-string by suppression symbols.

If the suppression symbols appear only to the left of the decimal point, any leading zero in the data which corresponds to a suppression symbol in the string is replaced by the replacement character. Suppression terminates at the first nonzero digit in the data represented by the suppression symbol string or at the decimal point, whichever is encountered first.

If all numeric character positions in the PICTURE character-string are represented by suppression symbols and the value of the data is not zero, the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero, the entire data item will be spaces if the suppression symbol is 'Z' or all '*', except for the actual decimal point, if the suppression symbol is '*'.

9. The symbols '+', '-', '*', 'Z', and the currency symbol, when used as floating replacement characters, are mutually exclusive within a given character-string.

Precedence Rules:

The following chart shows the order of precedence when using characters as symbols in a character-string. An 'X' at an intersection indicates that the symbol(s) at the top of the column may precede, in a given character-string, the symbol(s) at the left of the row. Arguments appearing in indicate that the symbols are mutually exclusive. The currency symbol is indicated by 'cs'.

In cases where the General Rules above and this chart conflict, the stated rules have precedence.

First Symbol / Second Symbol		Fixed Insertion								Other Symbols											
		B	0	,	.	{+ −}	{+ −}	{CR DB}	cs	A X	P	P	S	V	{Z *}	{Z *}	9	{+ −}	{+ −}	cs	cs
Fixed Insertion	B	X	X	X	X	X			X	X	X			X	X	X	X	X	X	X	X
	0	X	X	X	X	X			X	X	X		X	X	X	X	X	X	X	X	X
	,	X	X	X	X	X			X		X			X	X	X	X	X	X	X	X
	.	X	X	X		X			X		X				X		X	X		X	
	{+ −}										X										
	{+ −}	X	X	X	X				X		X			X	X	X	X			X	X
	{CR DB}	X	X	X	X				X		X			X	X	X	X			X	X
	cs					X					X			X							
Other Symbols	A X	X	X							X							X				
	P										X		X	X							
	P	X	X	X		X	X	X	X			X	X		X		X	X		X	
	S																				
	V	X	X	X		X			X			X	X		X		X	X		X	
	{Z *}	X	X	X		X			X						X						
	{Z *}	X	X	X	X	X			X		X			X	X	X					
	9	X	X	X	X	X			X	X	X		X	X	X		X	X		X	
	{+ −}	X	X	X					X									X			
	{+ −}	X	X	X	X				X		X			X				X	X		
	cs	X	X	X		X														X	
	cs	X	X	X	X	X					X			X						X	X

At least one of the symbols 'A', 'X', 'Z', '9', or '*', or at least two of the symbols '+', '-', or 'cs' must be present in a PICTURE string.

Fixed insertion symbols '+' and '-', and other symbol 'P' appear twice. The first occurrence represents their use to the left of the PICTURE's numeric character positions and the second their use to the right of the PICTURE's numeric character positions. Other symbols 'Z', '*', 'cs', '+', and '-' appear twice. The first occurrence represents the use before the decimal point position; the second the use after the decimal point position.

The RECORD CONTAINS clause in an FD or SD entry is used to specify the size of data records.

General Format:

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

Syntax Rule:

1. Integer-1 and integer-2 must be unsigned nonzero integers.

General Rules:

1. The size of each data record is completely defined in the record description (data description) entry; therefore, this clause is never required. When the RECORD CONTAINS clause is used, however, the following rules apply:

   a. Integer-2 may not be used by itself unless all of the data records in the file are the same size. In this case, integer-2 represents the exact number of characters in the data record. When both integer-1 and integer-2 are given, they refer to the minimum number of characters in the smallest data record and the maximum number of characters in the largest data record, respectively.

   b. The size of the data record is specified in terms of the number of characters in the standard data format contained in the logical record, regardless of the type of characters used to represent the items within the logical record. The size of the record is determined by adding the number of characters in all the fixed-length elementary items to the sum of the maximum number of characters in all variable-length items subordinate to the record. This sum may be different from the actual size of the record. Refer to the SYNCHRONIZED and USAGE clauses.

The RECORDING MODE clause in a file description entry is used to specify the format or organization of data on magnetic tape.

General Format:

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

Syntax Rule:

1. Whenever an APPLY SYSTEM STANDARD phrase in the I-O-CONTROL paragraph is specified for a file, the RECORDING MODE clause, if one is given for the file, must specify BINARY HIGH density.

General Rules:

1. If any data item or report item associated with a file is of any usage other than USAGE DISPLAY, the BCD option must not be used.
2. If the RECORDING MODE clause is omitted, the file will be assumed to be recorded in the binary high density mode.
3. Whenever either the APPLY SERIAL phrase or the APPLY VLR phrase in the I-O-CONTROL paragraph is specified for a file, the BCD option must not be used.

The REDEFINES clause is used to describe the same memory area by different data description entries. This is accomplished in either the File Section or Working-Storage Section; it allows the same memory area to contain different data items.

General Format:

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

NOTE: Level-number and data-name-1 are not part of the REDEFINES clause; they are shown only for clarity.

Syntax Rules:

1. The REDEFINES clause must immediately follow data-name-1.
2. The level-numbers of data-name-1 and data-name-2 must be identical but cannot be 66 or 88.
3. The REDEFINES clause cannot be used with level 01 data-names in the File Section. Implicit redefinition is provided by the DATA RECORDS clause in the file description entry.

General Rules:

1. Redefinition begins at data-name-2 and continues until a level-number less than or equal to that of data-name-2 is encountered.
2. When the level-number of data-name-1 is other than 01, then data-name-1 must specify a memory area of the same size as data-name-2. The REDEFINES clause specifies only the redefinition of a memory area, not of the data items that occupy the area.
3. The same memory area may be redefined as many times as required. The entries giving the new descriptions of the memory area must follow the entries defining the area being redefined, without entries defining new memory areas intervening. Multiple redefinitions of the same memory area must all use as data-name-2 the data-name of the entry that originally defined the area.
4. The data description entry for data-name-2 cannot contain an OCCURS clause nor can data-name-2 be subordinate to an entry containing an OCCURS clause. Neither the original definition nor any subsequent redefinitions of the area can include an item whose size is variable as defined for the OCCURS clause.
5. The entries giving the new description of the memory area must not contain VALUE clauses, except in condition-name entries.

6. Use caution when specifying REDEFINES of a noncontiguous data item (level 77) since such data items are implicitly synchronized based on the class.

7. When the REDEFINES clause is specified in the Working-Storage Section and more than fifty noncontiguous data items (level 77) are defined, the REDEFINES clause and the item it redefines must be included in the same group of fifty items; that is, in the first fifty level 77 items, or the second fifty level 77 items, etc.

The RENAMES clause is a level 66 data description entry that permits alternative, possibly overlapping, groupings of elementary items.

General Format:

```
66 data-name-1 RENAMES data-name-2  [ { THRU      } data-name-3 ].
                                      { THROUGH   }
```

NOTE: Level-number 66 and data-name-1 are not part of the RENAMES clause; they are shown only for clarity.

Syntax Rules:

1. All RENAMES entries associated with a given logical record must immediately follow that record's last data description entry.

2. Data-name-2 and data-name-3 must be names of elementary items or groups of elementary items in the associated logical record. They cannot, however, be the same data-name.

3. A level 66 entry may not be used to rename another level 66 entry nor may it be used to rename a level 01, 77, or 88 entry.

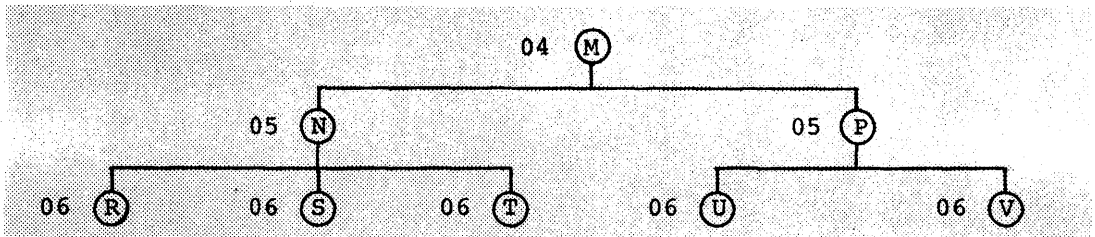
4. Data-name-1 may not be used as a qualifier, and may only be qualified by the names of the level 01, FD, or SD entries.

5. Neither data-name-2 nor data-name-3 may have an OCCURS clause in its data description entry nor be subordinate to an item having an OCCURS clause in its data description entry.

6. Data-name-2 and data-name-3 may be qualified.

7. The words THRU and THROUGH are equivalent.

8. The beginning of the area described by data-name-3 may not be to the left of the beginning of the area described by data-name-2. The end of the area described by data-name-3 must be to the right of the end of the area described by data-name-2. Data-name-3, therefore, may not be subordinate to data-name-2. The following diagram illustrates this rule.




```
Correct usage:    66 Z RENAMES R THRU T.

Incorrect usage:  66 Z RENAMES N THRU T.
```

General Rules:

1. One or more RENAMES entries may be written for a logical record.

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

3. When data-name-3 is not specified, data-name-2 can be either a group or an elementary item. When data-name-2 is a group item, data-name-1 is treated as a group item. When data-name-2 is an elementary item, data-name-1 is treated as an elementary item.

4. When data-name-3 is specified, none of the elementary items within the range, including data-name-2 and data-name-3, can be of variable length.

The REPORT(S) clause in a file description entry is used to cross-reference the report description entries with their associated file description entries.

General Format:

$$\left\{\begin{matrix}\underline{\text{REPORT}}\ \text{IS} \\ \underline{\text{REPORTS}}\ \text{ARE}\end{matrix}\right\}\ \text{report-name-1}\ \left[,\ \text{report-name-2}\right]\ \ldots$$

Syntax Rule:

1. Each report-name listed in a file description entry must be the subject of a report description entry in the Report Section.

General Rules:

1. The REPORT clause is required in the file description entry if the file being described is an output report file or is to contain output report records.

2. The presence of more than one report-name indicates that the file contains more than one report. These reports may be of different sizes, formats, etc. The order in which the report-names are listed is not significant.

The RESET clause in a report group description entry refers to the identifier used in the CONTROL clause of the associated RD entry that causes the SUM counter in the elementary item entry to be reset to zero on a control break.

General Format:

$$\underline{\text{RESET}}\ \text{ON}\ \left\{\begin{array}{l}\text{identifier-1}\\ \underline{\text{FINAL}}\end{array}\right\}$$

Syntax Rules:

1. Identifier-1 must be one of the identifiers described in the CONTROL clause in the report description entry. Identifier-1 must be a higher level CONTROL clause identifier than the CONTROL clause identifier associated with the CONTROL FOOTING report group in which the SUM and RESET clauses appear.

2. The RESET clause may be used only in conjunction with a SUM clause at the elementary level.

General Rules:

1. After presentation of the TYPE CONTROL FOOTING report group, the counters associated with the report group are reset to zero unless an explicit RESET clause is given specifying reset based on a higher level control than the associated control for the report group.

2. The RESET clause may be used for progressive totaling of identifiers where subtotals of identifiers are desired without automatic reset ing upon producing the report group.

3. When FINAL is specified, the SUM counter is not reset to zero until the final control footing is produced at TERMINATE time.

SOURCE, SUM, VALUE

The SOURCE, SUM, and VALUE clauses in a report group description entry are used to define the purpose of the report item within the report group.

General Format:

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

Syntax Rules:

1. Each identifier must indicate an item appearing in the File Section or Working-Storage Section or must be the name of a SUM counter in the Report Section.
2. SOURCE (without SELECTED), SUM, and VALUE clauses can be used only at the elementary level. The SOURCE IS SELECTED clause can be used only at the group level.
3. When the SELECTED phrase is used, identifier-1 represents a group item. The identifiers described at the elementary level in the source record then become SOURCE entries in the associated report group. The SELECTED elementary level identifiers must be unique data-names.
4. Literal-1 may be numeric, nonnumeric, or a figurative constant.
5. Data-name-1 must not be qualified.

SOURCE Rules:

1. The SOURCE clause indicates a data item which is to be used as the source for this report item. This data item is called a SOURCE data item or a SOURCE item. The item is presented according to the PICTURE clause in the associated elementary report group description entry.
2. When the SELECTED phrase is specified, the elementary level items within identifier-1 are matched against the data-names specified at the elementary level within the report group. Matching data-names are selected as SOURCE item entries to be included and presented within the report group, according to the PICTURE and USAGE specifications given with the data-name in the report group description entry.

SOURCE, SUM, VALUE

SUM Rules:

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

2. If a SUM counter is referred to by a Procedure Division statement or Report Section entry, a data-name must be specified with the SUM clause entry. The data-name then represents the summation counter automatically generated by the Report Writer to total the operands specified immediately following the SUM. If a summation counter is never referred to, the counter need not be named explicitly by a data-name entry. A SUM counter is only algebraically incremented just before presentation of the TYPE DETAIL report group in which the item being summed appears as a SOURCE item.

3. Whether the SUM clause names the summation counter or not, the PICTURE clause must be specified for each SUM counter. Editing characters or editing clauses may be included in the description of a SUM counter. Editing of a SUM counter only occurs upon the presentation of that SUM counter. At all other times, the SUM counter is treated as a numeric data item. The SUM counter must be large enough to accommodate the summed quantity without truncation of integral digits.

4. Eacn item being summed, that is, identifier-2, identifier-3, etc., must appear as a SOURCE item in a TYPE DETAIL report group or be names of SUM counters in a TYPE CONTROL FOOTING report group at an equal or lower position in the control hierarchy. Although the items must be explicitly written in a TYPE DETAIL report group, they may actually be suppressed at presentation time. In this manner, direct association without ambiguity can be made from the current data available by a GENERATE statement to the data items to be presented within the Report Section.

5. If higher level report groups are indicated in the CONTROL hierarchy, counter updating procedures, commonly called 'rolling counters forward', take place prior to the reset operation.

6. The summation of data items defined as SUM counters in TYPE CONTROL FOOTING report groups is accomplished explicitly or implicitly with the Report Writer automatically handling the updating function. If a SUM CONTROL of a data item is not desired for presentation at a lower level but is desired for presentation at a higher level, the lower level SUM specification may be omitted. In this case, the same results are obtained as if the lower level SUM counter were specified.

7. The UPON data-name-1 phrase is required to obtain selective summation for a particular data item which is named as a SOURCE item in two or more TYPE DETAIL report groups. Identifier-2 and identifier-3 must be SOURCE data items in data-name-1. Data-name-1 must be the name of a TYPE DETAIL report group. If the UPON data-name-1 phrase is not used, identifier-2, identifier-3, etc., respectively, are added to the SUM counter at each execution of a GENERATE statement. This statement generates a TYPE DETAIL report group that contains the SUM operands at the elementary level.

   For further explanation, refer to the ADD statement.

SOURCE, SUM, VALUE

VALUE Rule:

1. The VALUE clause causes the report data item to assume the specified value each time its report group is presented.

The SYNCHRONIZED clause in a data description entry is used to specify the alignment of an elementary item with a computer word or words.

General Format:

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

Syntax Rules:

1. The SYNCHRONIZED clause can be used only with an elementary item.
2. SYNC is an abbreviation for SYNCHRONIZED.

General Rules:

1. 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 that can contain the item, with no part of any other item sharing those words.
2. The computer word, or words, containing the synchronized item may also have to contain some unused character positions in order to fill the computer word, or words. When SYNCHRONIZED LEFT is specified, these unused character positions (if any) will occupy the least significant portion of the last word of the data item. When SYNCHRONIZED RIGHT is specified, the unused positions (if any) will occupy the most significant portion of the first word of the data item. The unused character positions must not be described with FILLER items.
3. All unused character positions resulting from the SYNCHRONIZED clause appear in the external format.
4. Whenever a synchronized item is referenced in the source program, the original size of the item as shown in the PICTURE clause is used in determining any action which depends on size, such as justification or truncation. The REDEFINES clause, however, leads to a redefinition of a memory area, not just of the data items occupying the area. If SYNCHRONIZED clauses resulted in unused character positions in the original definition of the area, other than to the left of the first item being redefined, the new definition must account for all such character positions. If the first item in the original definition is SYNCHRONIZED RIGHT, the area being redefined begins in the leftmost character of the first word allocated to the original item. If the last item of the original definition is SYNCHRONIZED LEFT, the area being redefined extends to the rightmost character of the last word allocated to the original item.

5. When SYNCHRONIZED is specified for an item within the scope of an OCCURS clause, each occurrence of the item will be synchronized.

6. Data items described with USAGE COMPUTATIONAL or COMPUTATIONAL-n are automatically synchronized. Data items described with USAGE COMPUTATIONAL-4 are automatically synchronized on word and half-word boundaries. All other COMPUTATIONAL or COMPUTATIONAL-n data items are automatically synchronized on word boundaries.

7. If neither LEFT nor RIGHT is specified in the SYNCHRONIZED clause, the data items are implicitly synchronized as follows:

   a. Alphabetic and alphanumeric data items are SYNCHRONIZED LEFT.

   b. Numeric, numeric edited, and alphanumeric edited data items are SYNCHRONIZED RIGHT.

8. Unless otherwise specified in an explicitly stated SYNCHRONIZED clause, noncontiguous data items (level 77) in working-storage are implicitly synchronized as follows:

   a. Alphabetic and alphanumeric data items are SYNCHRONIZED LEFT.

   b. Numeric, numeric edited, and alphanumeric edited data items are SYNCHRONIZED RIGHT.

9. If the data description of an item contains the SYNCHRONIZED clause and an operational sign, the sign of the item will appear in the least significant character of the data item, regardless of whether the item is SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT.

The TYPE clause in a report group description entry is used to specify the particular type of report group that is described by this entry and to indicate the time at which the report group is to be generated.

General Format:

```
                  ┌                                                    ┐
                  │ REPORT HEADING                                     │
                  │ RH                                                 │
                  │ PAGE HEADING                                       │
                  │ PH                                                 │
                  │ OVERFLOW HEADING                                   │
                  │ OH                                                 │
                  │ {CONTROL HEADING} {identifier-n}                   │
                  │ {CH             } {FINAL     }                     │
TYPE IS           │ DETAIL                                             │
                  │ DE                                                 │
                  │ {CONTROL FOOTING} {identifier-n}                   │
                  │ {CF             } {FINAL     }                     │
                  │ OVERFLOW FOOTING                                   │
                  │ OV                                                 │
                  │ PAGE FOOTING                                       │
                  │ PF                                                 │
                  │ REPORT FOOTING                                     │
                  │ RF                                                 │
                  └                                                    ┘
```

Syntax Rule:

1. RH is an abbreviation for REPORT HEADING;
   PH is an abbreviation for PAGE HEADING;
   OH is an abbreviation for OVERFLOW HEADING;
   CH is an abbreviation for CONTROL HEADING;
   DE is an abbreviation for DETAIL;
   CF is an abbreviation for CONTROL FOOTING;
   OV is an abbreviation for OVERFLOW FOOTING;
   PF is an abbreviation for PAGE FOOTING;
   RF is an abbreviation for REPORT FOOTING.

General Rules:

1. The level-number 01 identifies a particular report group to be generated as output and the TYPE clause in this entry indicates the time for generation of this report group. If the report group is described as other than TYPE DETAIL, its generation is an automatic Report Writer function. If the report group is described with the TYPE DETAIL clause, the Procedure Division statement GENERATE data-name directs the Report Writer to produce the named report group.

2. The REPORT HEADING entry indicates a report group that is produced only once at the beginning of a report during the execution of the first GENERATE statement. There can be only one report group of this type in a report. SOURCE clauses used in TYPE RH report groups refer to the values of data items at the time the first GENERATE statement is executed.

3. The PAGE HEADING entry indicates a report group that is produced at the beginning of each page according to PAGE and OVERFLOW condition rules as specified in Rule 19. There can be only one report group of this type in a report.

4. The OVERFLOW HEADING entry indicates a report group that is produced at the beginning of a page following an OVERFLOW condition according to PAGE and OVERFLOW rules as specified in Rule 19. There can be only one report group of this type in a report.

5. The CONTROL HEADING entry indicates a report group that is produced at the beginning of a control group for a designated identifier or, in the case of FINAL, is produced once before the first control group at the initiation of a report during the execution of the first GENERATE statement. There can be only one report group of this type for each identifier and for the FINAL entry specified in a report. To produce CONTROL HEADING report groups, a control break must occur. SOURCE clauses used in TYPE CONTROL HEADING FINAL report groups refer to the values of the items at the time the first GENERATE statement is executed.

6. The DETAIL entry indicates a report group that is produced for each GENERATE statement in the Procedure Division. Each DETAIL report group must have a unique data-name at the 01 level in a report.

7. The CONTROL FOOTING entry indicates a report group that is produced at the end of a control group for a designated identifier or is produced once at the termination of a report ending a FINAL control group. There can be only one report group of this type for each identifier and for the FINAL entry specified in a report. To produce CONTROL FOOTING report groups, a control break must occur. SOURCE clauses used in TYPE CONTROL FOOTING FINAL report groups refer to the values of the items at the time the TERMINATE statement is executed.

8. The OVERFLOW FOOTING entry indicates a report group that is produced at the bottom of a page following an OVERFLOW condition according to PAGE and OVERFLOW rules as specified in Rule 19. There can be only one report group of this type in a report.

9. The PAGE FOOTING entry indicates a report group that is produced at the bottom of each page according to PAGE and OVERFLOW condition rules as specified in Rule 19. There can be only one report group of this type in a report.

10. The REPORT FOOTING entry indicates a report group that is produced only once at the termination of a report. There can be only one report group of this type in a report. SOURCE clauses used in TYPE REPORT FOOTING report groups refer to the values of the items at the time the TERMINATE statement is executed.

11. Identifiers, as well as FINAL, must be one of the identifiers described in the CONTROL(S) clause in the report description entry.

12. A FINAL type control break may be designated only once for CONTROL HEADING or CONTROL FOOTING entries within a report.

13. Nothing precedes a REPORT HEADING entry and nothing follows a REPORT FOOTING entry within a report.

14. The HEADING or FOOTING report groups occur in the following Report Writer sequence, if all exist for a given report:

```
REPORT HEADING (one occurrence only-first page)
PAGE HEADING or OVERFLOW HEADING
      .
      .
      .
CONTROL HEADING
DETAIL
CONTROL FOOTING
      .
      .
      .
PAGE FOOTING or OVERFLOW FOOTING
REPORT FOOTING (one occurrence only-last page)
```

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

```
Final Control Heading
Major Control Heading
      .
      .
      .
Minor Control Heading
```

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

```
Minor Control Footing
      .
      .
      .
Major Control Footing
Final Control Footing
```

16. CONTROL HEADING report groups appear with the current values of any indicated SOURCE data items before the DETAIL report groups of the CONTROL group are produced. CONTROL FOOTING report groups appear with the previous values of any indicated CONTROL SOURCE data items just after the DETAIL report groups of that CONTROL group have been produced. The USE procedures specified for a CONTROL FOOTING report group that refer to: a) SOURCE data items specified in the CONTROL(S) clause affect the previous value of the items; b) SOURCE data items not specified in the CONTROL(S) clause affect the current value of the items. These report groups appear whenever a control break is noted. LINE NUMBER determines the absolute or relative position of the CONTROL report groups exclusive of the other HEADING and FOOTING report groups.

17. The concept of the OVERFLOW condition in a Report Writer is based on the logical definition of a page format relative to the presentation of a complete control group. For purposes of the OVERFLOW condition, a complete control group depends on the change of a data item value within a designated order of specific data items. If the change is a minor control group break, the complete control group includes the HEADING, DETAIL, and FOOTING report groups associated with the minor control specification. If the change is a major control group break, the complete control group includes the HEADING, DETAIL, and FOOTING report groups associated with the minor, intermediate, and major control specifications. Thus, during process time, if a page format does not allow a complete control group to be presented within the definition of the page, an OVERFLOW condition is said to exist from the last DETAIL report group printed in the control group on one page to the first report group printed in the control group on the next page. Between the points of from and to described above, OVERFLOW FOOTING and OVERFLOW HEADING report groups may be produced, if specified. If a complete control group, as described above, and none of the next control groups can be presented within the definition of the page, a PAGE condition is said to exist from the last DETAIL report group and, therefore, PAGE FOOTING and PAGE HEADING report groups are produced, if specified.

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

19. To recognize the OVERFLOW condition within the Report Writer and to determine the difference between an OVERFLOW condition and a PAGE condition, the PAGE LIMIT(S) clause must be given, including an explicit LAST DETAIL phrase. If both TYPE PAGE HEADING and OVERFLOW HEADING or TYPE PAGE FOOTING and OVERFLOW FOOTING report groups are specified in the same report and if the LINE-COUNTER will exceed the LAST DETAIL limit for generation of the current report group, the following rules apply:

    a. Without an explicit PAGE LIMIT(S) FOOTING clause, if the current report group is not the first report group of a new control group, an OVERFLOW condition exists from this position on the page to the position on the next page where the FIRST DETAIL report group can be presented. If the current report group is the first report group of a new control group, a PAGE condition exists. TYPE CONTROL FOOTING report groups are considered part of the last control group. TYPE CONTROL HEADING report groups are considered part of the next or current control group.

TYPE

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

c. Without an explicit PAGE LIMIT(S) LAST DETAIL clause, an OVERFLOW condition cannot exist within the Report Writer. Therefore, with or without the PAGE LIMIT(S) FOOTING clause, TYPE PAGE FOOTING, as differentiated from TYPE OVERFLOW FOOTING, are the only report groups that are produced, if specified, after TYPE DETAIL and TYPE CONTROL report groups on a page.

d. The rules stated in Rule 18 above apply regardless of the conditions that may be recognized by the Report Writer as described in Rule 19.

The USAGE clause in a data description entry is used to indicate the dominant use of a data item or the manner in which a data item is represented in memory.

General Format:

```
[ USAGE IS ]  { COMPUTATIONAL   }
              { COMP            }
              { COMPUTATIONAL-1 }
              { COMP-1          }
              { COMPUTATIONAL-2 }
              { COMP-2          }
              { COMPUTATIONAL-3 }
              { COMP-3          }
              { COMPUTATIONAL-4 }
              { COMP-4          }
              { DISPLAY         }
              { DISPLAY-1       }
              { DISPLAY-2       }
              { INDEX           }
```

Syntax Rules:

1. COMP is an abbreviation for COMPUTATIONAL. COMP-n is an abbreviation for COMPUTATIONAL-n.
2. The PICTURE character-string for a COMPUTATIONAL or COMPUTATIONAL-n data item must only contain '9's, the operational sign 'S', and the assumed decimal point 'V'.
3. When USAGE COMPUTATIONAL-4 is specified, the object computer must be explicitly or implicitly 6000-EIS.
4. The only usage that can be specified for a report group description entry is USAGE DISPLAY.

General Rules:

1. If the usage of a data item is not specified, it is assumed to be USAGE DISPLAY.
2. The USAGE clause may be specified at any level of a hierarchical structure. If this clause is specified at a group level, it applies to all of the subordinate elementary items in the group and no subordinate item may specify a different usage.
3. The external format of a data item (as it is stored on a peripheral device) and its internal format (as it is stored in memory) are always the same.

4. When the USAGE clause is specified, the internal format invoked for a computational item must not conflict with the data characteristics specified for the item in the PICTURE clause. The USAGE clause permits a choice of the following internal formats for computational data items:

   a. COMPUTATIONAL represents decimal-precision binary. The data item is stored as a synchronized signed floating-point binary number. The PICTURE clause description must conform to the rules for numeric items. If the PICTURE clause specifies eight or less digits, the item is stored as a single-precision floating-point number; otherwise, it is stored as a double-precision floating-point number. To preserve fractional accuracy, each item is treated with a span multiplier as described in Section II of the COBOL User's Guide.

   b. COMPUTATIONAL-1 represents binary integer. The data item is stored as a synchronized signed fixed-point binary integer. The PICTURE clause description must conform to the rules for numeric items and the assumed decimal point must be immediately to the right of the rightmost digit position. If the PICTURE clause specifies eight or less digits, the item is stored as a single-precision binary integer; otherwise, it is stored as a double-precision binary integer.

   c. COMPUTATIONAL-2 represents floating-point binary. The data item is stored as a synchronized signed floating-point binary number. The PICTURE clause description must conform to the rules for numeric items. If the PICTURE clause specifies eight or less digits, the item is stored as a single-precision floating-point number; otherwise, it is stored as a double-precision floating-point number.

   d. COMPUTATIONAL-3 represents single-precision binary integer. The data item is stored as a synchronized signed single-precision fixed-point binary integer. The PICTURE clause description must conform to the rules for numeric items and the assumed decimal point must be immediately to the right of the rightmost digit position. The PICTURE clause may specify at most ten digits.

   e. COMPUTATIONAL-4 represents packed decimal. The data item is stored as a synchronized fixed-point packed decimal number. The PICTURE clause description must conform to the rules for numeric items. If the PICTURE character-string specifies an operational sign, the sign will be stored as a separate digit; that is, the data item will be one digit larger than the number of character positions described in the PICTURE character-string. If USAGE COMPUTATIONAL-4 is specified for a group item, the group item itself will be considered to be alphanumeric.

5. When the USAGE DISPLAY clause is specified, the data items are stored as standard data format characters. The USAGE clause permits a choice of the following internal formats for display data items:

   a. DISPLAY represents character-oriented data. The data item is stored in the native (Series 60/6000) six-bit character set. The PICTURE clause description may imply alphabetic, numeric, alphanumeric, or numeric edited data items.

b. DISPLAY-1 represents edited floating-point. The data item is stored in the native six-bit character set. The class of the item is implicitly alphanumeric, and it is formatted as a Numeric Representation Set 3 character-string as specified in the American National Standard for the Representation of Numeric Values in Character Strings for Information Interchange (X3.42-1975).

c. DISPLAY-2 represents character-oriented data. The data item is stored in a nonnative six-bit character set. The character set is the commercial collating character set described in Appendix D of the COBOL User's Guide.

The PICTURE clause description must imply a class of alphabetic or alphanumeric for the data item.

Although a DISPLAY-2 data item may not be compared to an item having any other USAGE, it may be compared to literals and figurative constants.

DISPLAY-2 data items must be explicitly moved to USAGE DISPLAY items if they are to appear on punched cards, printer listings, or similar external media. This function cannot be accomplished with a REDEFINES clause.

6. An elementary item described with the USAGE INDEX clause is called an index data item; the external and internal format of an index data item is single-precision binary integer. The elementary item described with the USAGE INDEX clause contains a value that must correspond to an occurrence number of a table element and must not be a conditional variable. It is assigned a single word of memory.

An index data item can be referenced directly only in a SEARCH statement, a SET statement, or in a relation condition. An index data item can be part of a group that is referenced in a MOVE statement or an input-output statement, in which case no conversion takes place.

The BLANK WHEN ZERO, JUSTIFIED, PICTURE, SYNCHRONIZED, and VALUE clauses must not be used to describe group items or elementary items described with the USAGE INDEX clause. If a group item is described with the USAGE INDEX clause, the elementary items in the group are all index data items. The group item itself is not an index data item and must not be used in SEARCH statements, SET statements, or in a relation condition.

7. Although the USAGE clause does not itself limit the use of the data item being described, some statements in the Procedure Division may restrict the USAGE clauses that may be applied to their operands. However, the USAGE clause may affect the decimal point or internal representation of the data item being described.

The VALUE clause in a data description entry is used to define the initial value of working-storage data items, or the values associated with a condition-name.

Format 1:

```
VALUE IS literal-1
```

Format 2:

```
88 condition-name-1 { VALUE IS  } literal-1
                    { VALUES ARE}

[ { THRU      } literal-2 ]  [ , literal-3  [ { THRU      } literal-4 ] ] ...
  { THROUGH   }              [               { THROUGH   }            ]
```

NOTE: Level-number 88 and condition-name-1 are not part of the VALUE clause; they are shown only for clarity.

Syntax Rules:

1. The words THRU and THROUGH are equivalent.
2. A signed numeric literal must be associated with a signed numeric PICTURE character-string.

General Rules:

1. The VALUE clause cannot be stated for any item whose size, explicitly or implicitly, is variable.
2. The VALUE clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the item. If the category of an elementary item is specified as numeric or alphabetic, it does not contradict the alphanumeric category of group items. The following rules apply:
   a. If the category of the item is numeric, all literals in the VALUE clause must be numeric literals. If the literal defines the value of a working-storage data item, the literal is aligned according to the alignment rules except that the literal must not have a value that would require truncation of nonzero digits. A negative numeric literal must be associated with a signed numeric (S9) PICTURE character-string.

b. If the category of the item is alphabetic, alphanumeric, alphanumeric edited, or numeric edited, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned in the data item according to the alignment rules (refer to the JUSTIFIED clause), except that the number of characters in the literal must not exceed the size of the item.

c. All numeric literals in a VALUE clause of an item must have a value which is within the range of values indicated by the PICTURE clause; for example, for PICTURE PPP99, the literal must be within the range .00000 through .00099.

d. The function of the BLANK WHEN ZERO clause or any editing characters in a PICTURE clause has no effect on the initialization of an item. The VALUE clause is the only clause that may (depending on its usage) provide initialization. Editing characters are included, however, in determining the size of the item. Therefore, the VALUE for an edited item must be presented in an edited form.

3. A figurative constant may be substituted in both Format 1 and Format 2 wherever a literal is specified.

4. The VALUE clause cannot be used for items whose USAGE IS INDEX.

Condition-Name Rules:

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

   a. Another condition-name.

   b. A level 66 item.

   c. A group containing items requiring separate handling due to synchronization, usage, etc.

2. In a condition-name entry, the VALUE clause is required. The VALUE clause and the condition-name itself are the only two clauses permitted in the entry. The characteristics of a condition-name are implicitly those of its conditional variable.

3. Format 2 can be used only in connection with condition-names. Whenever the THRU phrase is used, literal-1 must be less than literal-2, literal-3 less than literal-4, etc. No more than 24 ranges of values (THRU phrases) may be specified in a condition-name entry.

An example of condition-name entries follows:

```
03 GRADE PIC 9(2).

     88 PRIMARY VALUE IS 1.
     88 SECOND VALUE IS 2.
      .
      .
      .
     88 GRADE-SCHOOL VALUES ARE 1 THRU 6.
     88 JUNIOR-HIGH VALUES ARE 7 THRU 9.
     88 HIGH-SCHOOL VALUES ARE 10 THRU 12.
     88 GRADE-ERROR VALUES ARE 0, 13 THRU 99.
```

Data Description Entries Other Than Condition-Names:

1. Rules governing the use of the VALUE clause differ with the respective sections of the Data Division:

   a. In the File Section, the VALUE clause may be used only in condition-name entries.

   b. In the Working-Storage Section, the VALUE clause may be used in condition-name entries, and it may also be used to specify the initial value of any other data item; in which case, the clause causes the item to assume the specified value at the start of object program execution. If the VALUE clause is not used in an item's description, the initial value is undefined.

   c. In the Report Section, the VALUE clause causes the report data item to assume the specified value each time its report group is presented. This clause may be used only at the elementary level in the Report Section.

2. The VALUE clause must not be stated in a data description entry that contains an OCCURS clause, or in an entry that is subordinate to an entry containing an OCCURS clause. This rule does not apply to condition-name entries.

3. The VALUE clause must not be stated in a data description entry that contains a REDEFINES clause, or in an entry that is subordinate to an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.

4. If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal, and the group area is initialized without consideration for the individual elementary or group items contained within this group. The VALUE clause cannot be stated at the subordinate levels within this group.

5. The VALUE clause must not be written for a group containing items with descriptions including the JUSTIFIED, SYNCHRONIZED, or USAGE clauses (other than USAGE IS DISPLAY).

6. Within a given record description, the VALUE clause must not be stated in a data description entry that is subsequent to a data description entry in which an OCCURS clause with a DEPENDING ON phrase appears.

VALUE OF

The VALUE OF clause in a file description entry is used to particularize the description of an item in the label records associated with a file.

General Format:

```
VALUE OF  data-name-1   IS  { literal-1   }
                            { data-name-2 ▲}

        [ , data-name-3   IS  { literal-2   } ]...
                              { data-name-4▲} ]
```

Syntax Rule:

1. Data-name-1, data-name-2, data-name-3, etc., should be qualified when necessary, but cannot be subscripted or indexed, nor can they be items described with the USAGE IS INDEX clause.

General Rules:

1. Each data-name-1, data-name-3, etc., must be in one of the label records; ▲ data-name-2, ▲ data-name-4, etc., must be in the Working-Storage Section. For an:

   a. Input File: The appropriate label routine checks to see if the value of data-name-1 is equal to the value of literal-1, or of ▲ data-name-2, whichever has been specified.

   b. Output File: At the appropriate time, the value of data-name-1 is made equal to the value of literal-1, or of ▲ data-name-2, whichever has been specified.

2. A figurative constant may be substituted wherever a literal is specified.

3. If label records are standard (see LABEL RECORDS clause), then data-name-1, data-name-3, etc., must be one of the following:

   IDENTIFICATION or ID
   RETENTION-PERIOD

   a. When ID or IDENTIFICATION is used, a nonnumeric literal or a data-name that has a class of alphanumeric and size of no more than 12 characters must be associated with the fixed data-name.

   b. When RETENTION-PERIOD is used, a numeric literal not exceeding 999 must be specified. The value 999 signifies permanent retention.

4. For mass storage files, the VALUE OF IDENTIFICATION clause is ignored since a label record is not present on the external device. However, the standard label USE procedures are engaged at OPEN and CLOSE for such files.

SECTION VII

PROCEDURE DIVISION

DESCRIPTION OF THE PROCEDURE DIVISION

The Procedure Division contains the procedures required to solve a given problem. Procedures are written as sentences, combined to form paragraphs, which in turn may be combined to form sections. COBOL procedures are expressed in a manner similar to (but not identical with) ordinary English. The basic unit of procedure formation is a sentence or a group of successive sentences.

DECLARATIVES

Declarative sections must be grouped at the beginning of the Procedure Division preceded by the keyword DECLARATIVES and followed by the keywords END DECLARATIVES. The USE statement is called a declarative statement.

PROCEDURES

A procedure is composed of a paragraph, or group of successive paragraphs; or a section, or group of successive sections within the Procedure Division. If one paragraph is in a section, then all paragraphs must be in sections. A procedure-name is a word used to refer to a paragraph or section in the source program in which it occurs. It consists of a paragraph-name (which may be qualified) or a section-name.

The end of the Procedure Division and the physical end of the program is that physical position in a COBOL source program after which no further procedures appear.

A section consists of a section header followed by zero, one, or more successive paragraphs. A section header consists of a section-name followed by the required word SECTION, a priority-number if desired, and a period. A section ends immediately before the next section; at the end of the Procedure Division; or, in the declarative portion of the Procedure Division, at the keywords END DECLARATIVES.

A paragraph consists of a paragraph-name followed by zero, one, or more successive sentences. A paragraph ends immediately before the next paragraph-name or section-name; at the end of the Procedure Division; or, in the declaratives portion of the Procedure Division, at the keywords END DECLARATIVES.

A sentence consists of one or more statements and is terminated by a period followed by a space.

A statement is a syntactically valid combination of words and symbols beginning with a COBOL verb.

The term identifier is defined as the word or words necessary to make unique reference to a data item.

Execution begins with the first statement of the Procedure Division, excluding declaratives, or at an entry point. Statements are then executed in the order in which they are presented for compilation, except where the rules in this section indicate a different order.

STRUCTURE OF THE PROCEDURE DIVISION

Procedure Division Header

The Procedure Division is identified by and must begin with the following header:

```
PROCEDURE DIVISION.
```

Procedure Division Body

The body of the Procedure Division must conform to one of the following formats:

Format 1:

```
[ DECLARATIVES.

{ section-name  SECTION. declarative sentence

{ paragraph-name. {sentence }... } ... } ...

  END DECLARATIVES. ]

{ section-name  SECTION [priority-number]  .

{ paragraph-name. { sentence } ... } ... } ...
```

Format 2:

$$\left\{\text{paragraph-name.}\left\{\text{ sentence}\right\}\ \ldots\right\}\ \ldots$$

Procedure Division Segments

Priority-numbers may be used on section headers to provide for segmentation of an object program. They identify the fixed and independent segments of the program.

General Format:

section-name SECTION [priority-number].

Syntax Rules:

1. The priority-number must be an integer ranging in value from 0 through 99.
2. If the priority-number is omitted from the section header, the priority is assumed to be 0.

General Rules:

1. All sections which have the same priority-number constitute a program segment with that priority.
2. Segments with a priority-number of 0 through 49 belong to the fixed portion of the object program.
3. Segments with a priority-number of 50 through 99 are independent segments.
4. Sections in the declarative portion must not contain priority-numbers in their section headings. These sections are defined to have a priority of 0.

STATEMENTS AND SENTENCES

A statement is a syntactically valid combination of words and symbols beginning with a COBOL verb. The three types of statements are conditional statements, compiler-directing statements, and imperative-statements.

A sentence consists of a sequence of one or more statements, the last of which is terminated by a period. The three types of sentences are conditional sentences, compiler-directing sentences, and imperative sentences.

Conditional Statements and Sentences

A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value. A conditional statement is an IF, READ, SEARCH, or RETURN statement; a WRITE statement that specifies an INVALID KEY phrase; an arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) that specifies the SIZE ERROR phrase; or an ACCEPT MESSAGE statement that specifies the NO DATA phrase.

A conditional sentence is a conditional statement terminated by a period followed by a space. The conditional statement may be optionally preceded by an imperative-statement.

Compiler-Directing Statements and Sentences

A compiler-directing statement consists of a compiler-directing verb and its operands. The compiler-directing verbs are COPY, ENTER, and USE. A compiler-directing statement causes the compiler to take a specific action during compilation.

A compiler-directing statement terminated with a period is a compiler-directing sentence. For example:

```
USE AFTER ERROR PROCEDURE ON MASTER-FILE.
```

The keywords END PROGRAM or END OF PROGRAM may be considered as special compiler-directing statements. The END PROGRAM statement may be used to indicate the physical end of the source program. (A STOP RUN statement is used to indicate the logical end of the program.) Use of the END PROGRAM statement is recommended to facilitate the orderly completion of compilation of source programs that end with NOTE statements or comment lines. If specified, the END PROGRAM statement must begin in character position 8 (margin A of the reference format). It must also be terminated by a period (.).

Imperative-Statements and Sentences

An imperative-statement indicates a specific action to be taken by the object program. The imperative-statement is any statement that is neither a conditional statement nor a compiler-directing statement. An imperative-statement may consist of a sequence of imperative-statements, each possibly separated from the next by a separator. The imperative verbs are:

ACCEPT[3]	EXAMINE	PERFORM
ADD[1]	EXIT	RELEASE
ALTER	GENERATE	SEEK
CALL	GO TO	SET
CLOSE	INITIATE	SORT
COMPUTE[1]	MERGE	STOP
DISPLAY	MOVE	SUBTRACT[1]
DIVIDE[1]	MULTIPLY[1]	TERMINATE
	OPEN	WRITE[2]

Whenever the term 'imperative-statement' appears in the general format of statements described in this section, the 'imperative-statement' refers to that sequence of consecutive imperative-statements ended by a period, or an ELSE associated with a previous IF verb, or a WHEN associated with a previous SEARCH verb.

An imperative-statement terminated by a period is an imperative sentence. For example:

```
MOVE A TO B.
MOVE A TO B; ADD C TO D.
```

An imperative sentence may contain either a GO TO statement or a STOP RUN statement, which (if present) must be the last statement in the sentence. For example:

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

SENTENCE EXECUTION

In the following discussion, 'execution of a sentence or a statement within a sentence' means 'execution of object coding compiled from a sentence, or from a statement within a sentence which has been written in COBOL'. 'Transfer of control' means 'transfer of control in the object program by transferring (GOing) from one place to another place out of the written sequence'. 'Passing of control' means 'passing of control in the object program by passing from one place to the next place in the written sequence'.

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

---

[1] Without the optional phrase SIZE ERROR.
[2] Without the optional phrase INVALID KEY.
[3] Without the optional phrase NO DATA.

Conditional Sentence Execution

The general format of the conditional sentence is:

```
IF condition { statement-1      } { ELSE  statement-2      }
             { NEXT SENTENCE   } { [ELSE  NEXT SENTENCE]  } .
```

In the conditional sentence, 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 control is passed to the next sentence.

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

```
IF condition-1 imperative-statement-1 IF condition-2
statement-3 ELSE statement-4 ELSE statement-2
```

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

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

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

Compiler-Directing Sentence Execution

Compiler-directing sentences direct a COBOL compiler to take action at compilation time, rather than specifying action to be taken by the object program.

Imperative Sentence Execution

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

Control Relationship Between Procedures

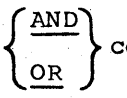
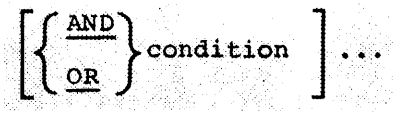
In COBOL, imperative and conditional sentences describe the procedure that is to be accomplished. The sentences are written successively, according to the rules of the reference format, to establish the sequence in which the object program is to execute the procedure.

Execution begins with the first statement after END DECLARATIVES if a declarative section is present. Statements are then executed in the order in which they are presented for compilation, except where the rules indicate some other order.

In executing procedures, control is transferred only to the beginning of a paragraph or section. 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.

CONDITIONS

A condition enables the object program to select between alternate paths of control, depending upon the truth value of a test. A condition is one of the following:

- Relation condition
- Sign condition
- Class condition
- Condition-name condition
- Switch-status condition
- NOT condition
- condition  condition 

Any condition may be enclosed in parentheses. The truth value of a parenthesized condition is determined from the evaluation of the truth values of its constituents. A parenthesized condition is a condition in the sense of the last two items of the preceding list.

Simple Conditions

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

A relation condition involves a comparison of two operands, each of which may be the data item referenced by an identifier, a literal, or the value resulting from an arithmetic-expression. The comparison of two literals is not permitted. Comparison of numeric operands is permitted regardless of their individual usages. All other comparisons require that the USAGE of the operands being compared is the same. If either of the operands is a group item, the nonnumeric comparison rules apply. The format for a relation condition is:

```
IF  {identifier-1       }  {IS [NOT]  >                  }  {identifier-2       }
    {literal-1          }  {IS [NOT]  <                  }  {literal-2          }
    {arithmetic-        }  {IS [NOT]  =                  }  {arithmetic-        }
    {  expression-1     }  {                             }  {  expression-2     }
                           {IS [NOT]  GREATER THAN        }
                           {IS [NOT]  LESS THAN           }
                           {IS [NOT]  EQUAL TO            }
                           {IS UNEQUAL TO                 }
                           {EQUALS                        }
                           {EXCEEDS                       }
```

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

In the preceding format, the first operand (identifier-1, literal-1, arithmetic-expression-1) is called the subject. The second operand (identifier-2, literal-2, arithmetic-expression-2) is called the object. The subject and the object cannot both be literals.

a. Comparison of Numeric Operands

For numeric operands, a comparison results in the determination that the algebraic value of one of the operands is less than, equal to, or greater than the other. The operand length, in terms of the number of digits, is not significant. Zero is considered to represent a unique value regardless of the length, sign, or implied decimal point location.

Comparison of these operands is permitted regardless of the manner in which their usage is described. Unsigned numeric operands are considered to be positive for comparison purposes.

b. Comparison of Nonnumeric Operands

For two nonnumeric operands, or one numeric (excluding the operational sign) and one nonnumeric operand, a comparison results in the determination that one of the operands is less than, equal to, or greater than the other with respect to an ordered character set. If one of the operands is specified as numeric, it must be an integer data item or an integer literal. Numeric and nonnumeric operands may be compared only when their usage is the same, explicitly or implicitly. Except when USAGE of the operands is DISPLAY-2, the character set order is determined by the Standard Collating Sequence. For DISPLAY-2 operands, the order is determined by the Commercial Collating Sequence. The collating sequences are presented in the COBOL User's Guide.

If the operands are of equal size, characters in corresponding character positions are compared 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 collating sequence. The operand which contains that character which is positioned higher in the collating sequence is determined to be the greater operand.

If the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by sufficient spaces to make the operands of equal size. If this process exhausts the characters of the operand of lesser size, then the operand of lesser size is less than the operand of larger size unless the remainder of the operand of larger size consists solely of spaces, in which case the two operands are equal.

c. Comparisons Involving Index-Names and/or Index Data Items

Full relation tests may be made between:

(1) Two index-names. The result is the same as if the corresponding occurrence numbers are compared.

(2) An index-name and a data item (other than an index data item) or a literal. The occurrence number that corresponds to the value of the index-name is compared to the data item or literal.

(3) An index data item and an index-name or another index data item. The actual values are compared without conversion.

The result of the comparison of an index data item with any data item or literal not specified in 1, 2, or 3 above is undefined.

SIGN CONDITION

The sign condition determines whether or not the algebraic value of an elementary numeric data item or an arithmetic-expression is less than, equal to, or greater than zero. The general format for a sign condition is:

```
IF { identifier            } IS [NOT] { POSITIVE }
   { arithmetic-expression }          { NEGATIVE }
                                      { ZERO     }
```

An operand is POSITIVE only if its value is greater than zero, NEGATIVE if its value is less than zero, and ZERO if its value is equal to zero. An operand whose value is zero is NOT POSITIVE and an operand whose value is zero is NOT NEGATIVE; the value zero is considered neither positive nor negative.

CLASS CONDITION

The class of any item can be tested as follows:

IF identifier IS [NOT] { NUMERIC / ALPHABETIC }

The usage of identifier must be explicitly or implicitly DISPLAY. The ALPHABETIC test cannot be used with an item whose data description describes the item as numeric. The item being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters 'A' through 'Z' and the space.

The NUMERIC test cannot be used with an item whose data description describes the item as alphabetic or as a group item composed of elementary items whose data description indicates the presence of operational sign(s). If the data description of the item being tested does not indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and an operational sign is not present.

CONDITION-NAME CONDITION

In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values associated with a condition-name in the Data Division. The general format for the condition-name condition is:

IF [NOT] condition-name

If the condition-name is associated with a range or ranges of values (that is, the VALUES ARE clause contains at least one 'literal THRU literal' phrase), then the conditional variable is tested to determine whether or not its value falls in this range, including the end values.

The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.

The result of the test is true if one of the values corresponding to the condition-name equals the value of its associated conditional variable.

SWITCH-STATUS CONDITION

In the SPECIAL-NAMES paragraph of the Environment Division, a condition-name may be associated with the ON or OFF status of a software switch. The switch is ON when its value is one, OFF when its value is zero. The status of such a switch may then be tested with a statement using the following format:

IF [NOT] condition-name

The results of this test are determined using the following table:

SPECIAL-NAMES Phrase	Switch Status	Test	
		IF condition-name	IF NOT condition-name
...ON STATUS IS condition-name	OFF	False	True
	ON	True	False
...OFF STATUS IS condition-name	OFF	True	False
	ON	False	True

Compound Conditions

Simple conditions can be combined with logical operators according to specified rules to form compound conditions. The logical operators AND, OR, and NOT must be preceded by a space and followed by a space. The meaning of the logical operators follows:

OR Logical Inclusive Or

AND Logical Conjunction

NOT Logical Negation

The general format of a compound condition is:

IF condition-1 { AND / OR } condition-2 [{ AND / OR } condition-n] ...

The word IF is shown to improve readability. Each condition can be either a relation condition, a sign condition, a class condition, a condition-name condition, or a switch-status condition.

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

Condition		Condition and Value		
A	B	NOT A	A AND B	A OR B
True	True	False	True	True
False	True	True	False	True
True	False	False	False	True
False	False	True	False	False

Thus, if A is true and B is false, the expression A AND B is false, while the expression A OR B is true.

The following table indicates the methods in which conditions and logical operators may be combined:

FIRST SYMBOL	SECOND SYMBOL					
	Condition	OR	AND	NOT	(	)
Condition	-	P	P	-	-	P
OR	P	-	-	P	P	-
AND	P	-	-	P	P	-
NOT	P[1]	-	-	-	P	-
(	P	-	-	P	P	-
)	-	P	P	-	-	P

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

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

1. If AND is the only logical connective used, then the compound condition is true if and only if each of the simple conditions is true.

2. If OR is the only logical connective used, then the compound condition is true if and only if one or more of the simple conditions is true.

---

[1] Permissible only if the condition itself does not contain a NOT.

3. If both AND and OR appear, then there are two cases to consider, depending on whether or not parentheses are used.

   a. Parentheses can be used to indicate grouping. They must always be paired, as in algebra, and the expressions within the parentheses will be evaluated first. The precedence of nested parenthetical expressions is the same as in algebra. That is, the innermost parenthetical expressions are evaluated first.

   b. If parentheses are not used, then the conditions are grouped first according to AND, proceeding from left to right, and then by OR, proceeding from left to right. That is, the logical operator AND has precedence over the logical operator OR in the same sense that the arithmetic operator * (multiplication) has precedence over the arithmetic operator + (addition).

4. When NOT precedes a parenthesized condition, it reverses the logical value of the parenthesized condition; that is, NOT (condition) is true when (condition) is false. For example, NOT (A AND B) is true if either A or B is false. Thus, NOT (A AND B) is equivalent to NOT A OR NOT B, and is true when A and B are not both true (i.e., when either is false or both are false). Similarly, NOT (A OR B) is equivalent to NOT A AND NOT B, and is true only when A and B are both false.

Abbreviated Combined Relation Conditions

Only conditions involving full relation tests have three terms (a subject, a relation, and an object). To simplify writing lengthy expressions, COBOL allows the omission of some of these terms in certain forms of compound conditions.

When relation conditions are written in a consecutive sequence, any relation condition except the first may be abbreviated by:

1. Omitting the subject of the relation condition, or

2. Omitting the subject and relational operator of the relation condition.

Within a sequence of relation conditions, both forms of abbreviations may be used. The effect of using them is as if the omitted subject were replaced by the last preceding stated subject or the omitted relational operator were replaced by the last preceding stated relational operator.

Ambiguity may result from using 'NOT' in conjunction with abbreviations. In this event, NOT is interpreted as a logical operator rather than as part of a relational operation. Thus:

```
a > b AND NOT > c OR d
```

is equivalent to:

```
a >b AND NOT a > c OR a > d     or

a >b AND (NOT a > c) OR a > d
```

Use of the NOT Operator

Simple IF sentences may be preferred when making a conditional test to avoid the possibility of misusing the NOT logical operator and to interpret the source language more clearly. When the NOT logical operator is used in IF sentences, it must precede a left parenthesis or a simple condition which does not contain a 'NOT'.

Evaluation Rules for Conditions

The evaluation rules for conditions are similar to those given for arithmetic-expressions except that the following hierarchy applies:

- Arithmetic-expression
- All relational operators
- NOT
- AND
- OR

ARITHMETIC-EXPRESSIONS

An arithmetic-expression can be an identifier described as a numeric elementary item; a numeric literal; such identifiers and literals separated by arithmetic operators; two arithmetic-expressions separated by an arithmetic operator; or an arithmetic-expression enclosed by parentheses. Any arithmetic-expression may be preceded by a unary operator. The permissible combinations of variables, numeric literals, arithmetic operators, and parentheses are presented in the table contained in the Formation and Evaluation Rules paragraph below.

Identifiers and literals appearing in an arithmetic-expression must represent either numeric elementary items or numeric literals on which arithmetic may be performed.

Arithmetic Operators

Five binary arithmetic operators and two unary arithmetic operators may be used in arithmetic-expressions. They are represented by specific characters which must be preceded by a space and followed by a space.

Binary Arithmetic Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Unary Arithmetic Operator	Meaning
+	The effect of multiplication by the numeric literal +1
-	The effect of multiplication by the numeric literal -1

Formation and Evaluation Rules for Arithmetic-Expressions

The formation and evaluation rules for arithmetic-expressions are presented below.

1. Parentheses may be used in arithmetic-expressions to specify the order in which elements are to be evaluated. When parentheses are used, a space is allowed between the left parenthesis and the leftmost element and between the right parenthesis and the rightmost element, if desired. Expressions within parentheses are evaluated first and, within nested parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of execution is implied:

   - Unary Plus and Minus
   - Exponentiation
   - Multiplication and Division
   - Addition and Subtraction

2. Parentheses are employed either to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear, or to modify the normal hierarchical sequence of execution in expressions where some deviation from the normal precedence is required.

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

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

   The following usages of exponentiation are not allowed and may produce unpredictable results:

   - The value zero exponentiated by the value zero.
   - The value zero exponentiated by a negative value.
   - A negative value exponentiated by a nonintegral value.

3. The methods in which operators, variables, and parentheses may be combined in an arithmetic-expression are summarized in the following table.

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

- The letter 'P' indicates a permissible pair of symbols.
- The character '-' represents an invalid pair.
- 'VARIABLE' represents an identifier or literal.

4. An arithmetic-expression may begin only with the symbol '(', '+', '-', or a variable, and may end only with a ')' or a variable. There must be a one-to-one correspondence between left and right parentheses of an arithmetic-expression so that each left parenthesis is to the left of its corresponding right parenthesis.

5. Arithmetic-expressions allow the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items. (See Arithmetic Statements.)

COMMON OPTIONS IN STATEMENT FORMATS

In the statement descriptions of the Procedure Division, several options appear frequently: the ROUNDED option, the SIZE ERROR option, and the CORRESPONDING option.

In the discussion below, a resultant-identifier is that identifier associated with a result of an arithmetic operation.

ROUNDED Option

If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant-identifier, truncation is relative to the size provided for the resultant-identifier. When rounding is requested, the absolute value of the resultant-identifier is increased by one (1) whenever the most significant digit of the excess is greater than or equal to five (5).

The following shows the effect of specifying the ROUNDED option.

Result of Arithmetic Operation	PICTURE of Resultant-Identifier	Values Stored in Resultant-Identifier
3.14	S9V9	3.1
3.15	S9V9	3.2
-3.14	S9V9	-3.1
-3.15	S9V9	-3.2

When the low-order integer positions in a resultant-identifier are represented by the character 'P' in the PICTURE for that resultant-identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

SIZE ERROR Option

If, after decimal point alignment, the value of the result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. Division by zero always causes a size error condition. The size error condition applies only to the final results of an arithmetic operation and not to intermediate results; except in the MULTIPLY and DIVIDE statements, in which case the size error condition applies to the intermediate results as well. If the ROUNDED option is specified, rounding takes place before checking for size error. When such a size error condition occurs, the subsequent action depends on whether or not the SIZE ERROR option is specified.

1. If the SIZE ERROR option is not specified and a size error condition occurs, the value of those resultant-identifier(s) affected is undefined. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation.

2. If the SIZE ERROR option is specified and a size error condition occurs, then the prior values of resultant-identifier(s) affected by the size errors are not altered. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation. After completion of the execution of this operation, the imperative-statement in the SIZE ERROR option is executed.

   For ADD and SUBTRACT CORRESPONDING, if any of the individual operations produce a size error condition, the imperative-statement in the SIZE ERROR phrase is not executed until all of the individual additions or subtractions are completed.

CORRESPONDING Option

For the purpose of this discussion, d1 and d2 represent identifiers that refer to group items. A pair of data items, one from d1 and one from d2, correspond if the following conditions exist:

1. A data item in d1 and a data item in d2 have the same name and the same qualification up to, but not including, d1 and d2.

2. At least one of the data items is an elementary data item in the case of a MOVE statement with the CORRESPONDING option; or both of the data items are elementary numeric data items in the case of the ADD or SUBTRACT statements with the CORRESPONDING option.

3. Neither d1 nor d2 may be data items with level-number 66, 77, or 88 nor be described with the USAGE IS INDEX clause.

4. A data item that is subordinate to d1 or d2 and contains a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause is ignored, as well as those data items subordinate to the data item that contains the REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause. However, d1 and d2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses.

Arithmetic Statements

The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements. They have several common rules.

1. All literals used in arithmetic statements must be numeric.

2. The data description of each identifier used as an operand must be that of an elementary numeric item.

3. The data descriptions of the operands need not be the same; any necessary conversion, format transformation, and/or decimal point alignment is supplied throughout the calculation.

4. The maximum size of each operand is 18 decimal digits. The composite of operands (a hypothetical data item resulting from the superimposition of specified operands in a given statement, aligned on their decimal points) must not contain more than 18 digits. The compiler ensures that enough places are carried so that significant digits are not lost during the calculation of intermediate results.

5. Editing symbols must not be specified in the descriptions of any operand, except in a resultant item that only receives the calculated result but is not used in the computation itself.

   The resultant item of a COMPUTE statement may be an edited item. The resultant of an ADD, SUBTRACT, MULTIPLY, or DIVIDE statement may be an edited item only when the GIVING option is specified. Operands in a computation must not be edited items in any other circumstances.

6. When the number of decimal places in a result is greater than the number of decimal places associated with the resultant-identifier, truncation occurs. However, when the ROUNDED option is specified for a resultant-identifier, the least significant digit of the resultant-identifier is increased by 1 when the most significant digit of the truncated excess is equal to or greater than 5.

7. A size error occurs when the magnitude of the calculated result exceeds the largest magnitude that can be contained in the resultant-identifier. When a size error occurs and the ON SIZE ERROR option is specified, the value of the resultant-identifier is not altered and the imperative-statement is executed.

Overlapping Operands

When a sending and a receiving item in an arithmetic statement or in an EXAMINE, MOVE, or SET statement share a part of their storage areas, the result of the execution of such a statement is undefined.

Multiple Results in Arithmetic Statements

The ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements may have multiple results. Such statements behave as though they had been written as:

1. Statements which perform all arithmetic necessary to arrive at the result to be stored in the receiving items, and store that result in a temporary storage location.

2. A sequence of statements transferring or combining the value of this temporary location with a single result. These statements are considered to be written in the same left-to-right sequence in which the multiple results are listed.

   The result of the statement

   ```
   ADD a, b, c TO c, d (c), e
   ```

   is equivalent to:

   ```
   ADD a, b, c GIVING temp
   ADD temp TO c
   ADD temp TO d (c)
   ADD temp TO e
   ```

   where 'temp' is an intermediate result item provided by the compiler.

CATEGORIES OF VERBS

Verbs available in Series 60/6000 COBOL are listed below within their functional categories.

Category	Verb
Arithmetic	ADD COMPUTE DIVIDE EXAMINE (TALLYING) MULTIPLY SUBTRACT
Compiler-Directing	COPY ENTER USE

Category	Verb
Conditional	ADD (SIZE ERROR) ACCEPT MESSAGE (NO DATA) COMPUTE (SIZE ERROR) DIVIDE (SIZE ERROR) GO TO (DEPENDING) IF MULTIPLY (SIZE ERROR) READ (END or INVALID KEY) RETURN (END) SEARCH SUBTRACT (SIZE ERROR) WRITE (INVALID KEY)
Data Movement	EXAMINE (REPLACING) MOVE
Ending	STOP
Input-Output	ACCEPT ACCEPT MESSAGE CLOSE DISPLAY OPEN READ SEEK STOP (literal) WRITE
Inter-Program Communicating	CALL
Procedure Branching	ALTER CALL EXIT GO TO PERFORM
Report Writing	GENERATE INITIATE TERMINATE
Ordering	MERGE RELEASE RETURN SORT
Table Handling	SEARCH SET

NOTE: IF is a verb only in the COBOL sense; it is recognized that it is not an English verb.

SPECIFIC STATEMENT FORMATS

The specific statement formats, with associated restrictions and limitations, are contained on the following pages in alphabetic sequence.

The ACCEPT statement is used to cause low-volume data to be made available to the specified identifier from sources described in the SPECIAL-NAMES paragraph.

Format 1:

```
ACCEPT identifier [FROM mnemonic-name]
```

Format 2:

```
ACCEPT MESSAGE identifier FROM mnemonic-name
       [NO DATA imperative-statement]
```

Syntax Rules:

1. The mnemonic-name must be associated with one of the options defined in the SPECIAL-NAMES paragraph.
2. The identifier must be a level 01 or a level 77 item in working-storage.

General Rules:

Format 1:

1. An ACCEPT statement may be used to obtain input data from any of the following sources:
   a. GIN (the system input feature of the operating system).
   b. REMOTE (a terminal not operating under the control of the Transaction Processing System).
   c. GLAPS (an operating system feature that provides accumulated processor time for the current run unit).
   d. GTIME (an operating system feature that provides the system date and the system clock time. If the HMS option is specified, the time is given in hours, minutes, and seconds).
   e. CONSOLE and TYPEWRITER (the system operator interface).
   f. SWITCH (a portion of the program switch word, a special software feature provided by the operating system).

   When the FROM mnemonic-name phrase is not used, the input source is considered to be system input (GIN). The FROM phrase must be specified for any other input source, and the mnemonic-name must be a user-supplied word associated with an input source by a phrase in the SPECIAL-NAMES paragraph.

An ACCEPT statement may not be executed, in an attempt to obtain data from system input (GIN), as a part of a label procedure appearing in the declarative portion of a program.

2. When utilizing the system input file (GIN) via ACCEPT statements, the user may either omit the FROM phrase or associate a mnemonic-name with GIN in the SPECIAL-NAMES paragraph. The data item referenced in the ACCEPT statement must be described with USAGE DISPLAY, explicitly or implicitly. Each record of the system input file is assumed to be a Hollerith card image in which the data occupies the leftmost character positions. No automatic format check or conversion is provided, so it is recommended that the user employ IF statements to assure that the input card contents satisfy the receiving item's description. Similarly, no automatic end-of-file provision is available, so the user must provide an end-of-file test if the volume of system input data can vary. However, each ACCEPT statement executed after the system input is exhausted obtains all spaces, as if a blank card had been read.

3. The utilization of ACCEPT statements that receive data from GIN in a module overlay environment must be carefully planned to avoid possible overlay loading on top of the COBOL subroutine that controls input from GIN. One method that may be used to avoid such an overlay is to place at least one ACCEPT statement in the main module that will never be overlayed. That statement need not actually be executed.

4. Each ACCEPT statement whose mnemonic-name is associated with REMOTE will cause a single interaction with the remote terminal. The terminal operator is notified of the need for a response by a carriage return followed by the display of the character '?' and the ringing of the terminal's bell, if the terminal is so equipped. The input characters, if any, are converted to Hollerith, USAGE DISPLAY characters and are moved into the data item referenced in the ACCEPT statement. The referenced data item should be described, explicitly or implicitly, with USAGE DISPLAY. No automatic format check or radix conversion is performed.

   If a remote terminal is represented by a mnemonic-name in the ACCEPT and DISPLAY statements, connections must be made to the appropriate device at object program execution. A connection may be obtained:

   a. By using a 'talk' mode in the time sharing environment. Refer to the TSS Terminal/Batch Interface manual and the TSS System Programmers' Reference Manual.

   b. By obtaining a connection through an external communications system. Refer to the Network Processing Supervisor (NPS) manual or to the Remote Terminal Supervisor (GRTS) manual.

   c. By requesting a line switch from the Transaction Processing Executive (TPE).

5. Each ACCEPT statement whose mnemonic-name is associated with GLAPS will cause the identifier to receive the processor time accumulated by the current run unit. Time resolution is given in units of 1/64 millisecond. The receiving identifier data item must be a working-storage data item whose description is equivalent to the following:

```
77  data-name  PICTURE 9(10)  USAGE COMPUTATIONAL-3.
```

6. Each ACCEPT statement whose mnemonic-name is associated with GTIME, without the HMS option, will cause the identifier to receive the current system date and system clock time. Time resolution is given in units of 1/64 millisecond. The receiving identifier must be a working-storage data item whose description is equivalent to the following:

```
01  data-name.
    02  MONTH  PICTURE  99.
    02  DAY-OF-MONTH  PICTURE  99.
    02  YEAR  PICTURE  99.
    02  TYME  PICTURE  9(10)  USAGE COMPUTATIONAL-3.
```

7. If the HMS option is specified in the GTIME phrase to obtain time resolution in terms of hours, minutes, and seconds, the receiving identifier must be described with USAGE DISPLAY (explicitly or implicitly). For example, the receiving identifier could be described as:

```
01  data-name.
    02  MONTH  PICTURE 99.
    02  DAY-OF-MONTH  PICTURE 99.
    02  YEAR  PICTURE 99.
    02  TYME  PICTURE 9(6).
```

NOTE: The data-names used in these examples are for illustration only.

8. If both forms of the GTIME phrase (with HMS and without HMS) are specified within the same COBOL program, the HMS option overrides and all GTIME requests are returned in hours, minutes, and seconds.

9. Each ACCEPT statement whose mnemonic-name is associated with CONSOLE will cause a single interaction with the operator's console. The system operator is notified that a response is expected by the display of a message. The message will have one of the following forms:

   a. If a DISPLAY statement associated with CONSOLE has been executed, the message will be the text of the last line associated with the latest prior DISPLAY statement associated with CONSOLE. The message will be followed by the characters '???'.

   b. If no prior DISPLAY statement associated with CONSOLE has been executed, the message will be 'TYPEIN EXPECTED...'.

The input characters, if any, are treated as Hollerith, USAGE DISPLAY characters and are moved into the data item referenced by the ACCEPT statement. The referenced data item should be described, explicitly or implicitly, with USAGE DISPLAY. No automatic format check or radix conversion is performed.

10. Each ACCEPT statement whose mnemonic-name is associated with TYPEWRITER will cause a single interaction with the operator's console. The system operator is notified that a response is expected by a carriage return followed by the message 'TYPEIN EXPECTED...'. The input characters, if any, are treated as Hollerith, USAGE DISPLAY characters and are moved into the data item referenced in the ACCEPT statement. The referenced data item should be described, explicitly or implicitly, with USAGE DISPLAY. No automatic format check or radix conversion is performed. System console input-output is not recommended unless very unusual circumstances exist.

11. If a mnemonic-name associated with SWITCH is specified, the ACCEPT statement causes the value of the identifier to be set to 1 if the switch is ON, or set to 0 if the switch is OFF. The identifier must be a data item in the Working-Storage Section whose description is equivalent to the following:

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

12. Refer to Section VI of the COBOL User's Guide for additional information.

Format 2:

1. In Format 2, mnemonic-name must be associated with COMMUNICATION-DEVICE in the SPECIAL-NAMES paragraph. The ACCEPT MESSAGE statement is used to obtain a message from the transaction processing intercom facility.

2. If the user has specified NO DATA in the ACCEPT MESSAGE statement, the specified imperative-statement is executed when no data is available.

3. If the NO DATA phrase in a Transaction Processing Applications Program (TPAP) is omitted, control is returned to the statement immediately following the ACCEPT MESSAGE statement when data is received. However, if no data is available, the program is suspended until data is available and, at that time, control is returned to the statement following the ACCEPT MESSAGE statement.

4. When data is available, it is moved to the internal work area specified in the ACCEPT MESSAGE statement. The message is transferred to the receiving area (referenced by identifier) aligned to the left without space-fill.

5. After a transaction request is received, the input header information for the current message may be referenced. Refer to Section VII of the COBOL User's Guide.

The ADD statement is used to sum two or more numeric operands and store the result.

Format 1:

```
ADD  {literal-1   }  [, literal-2   ]
     {identifier-1}  [, identifier-2]  ... TO identifier-m [ROUNDED]
     [, identifier-n [ROUNDED]]  ...
        [ON SIZE ERROR imperative-statement]
```

Format 2:

```
ADD  {literal-1   }  {, literal-2   }  [, literal-3   ]  ...
     {identifier-1}  {, identifier-2}  [, identifier-3]
     GIVING identifier-m [ROUNDED]
     [, identifier-n [ROUNDED]] ...
        [ON SIZE ERROR imperative-statement]
```

Format 3:

```
ADD  {CORR          }  identifier-1 TO identifier-2 [ROUNDED]
     {CORRESPONDING }
        [ON SIZE ERROR imperative-statement]
```

Syntax Rules:

1. When Format 1 or Format 2 is used, each identifier must refer to an elementary numeric item, except that in Format 2 the identifier following the word GIVING must refer either to an elementary numeric item or to an elementary numeric edited item.

2. Each literal must be a numeric literal.

3. No literal or identifier may exceed 18 decimal digits in size. The composite of operands (the hypothetical data item resulting from the superimposition of all operands of a given statement, excluding the data items that follow the word GIVING, aligned on their decimal points) must not contain more than 18 digits.

4. CORR is an abbreviation for CORRESPONDING.

General Rules:

1. When Format 1 is used, the values of the operands preceding the word TO are added together. That sum is then added to the current value of each identifier-m, identifier-n, ..., and the result is stored in each resultant-identifier: identifier-m, identifier-n, ..., respectively.

2. When Format 2 is used, the values of the operands preceding the word GIVING are added together. That sum is then stored as the new value of each identifier-m, identifier-n, ..., which are the resultant-identifiers.

3. When Format 3 is used, the data items in identifier-1 are added to and stored in corresponding data items in identifier-2.

4. Refer to the Common Options in Statement Formats paragraph in this section for an explanation of the uses of ROUNDED, SIZE ERROR, CORRESPONDING, and multiple results in arithmetic statements.

The ALTER statement is used to change the destination of a GO TO statement from one procedure-name to another.

General Format:

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

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

Syntax Rules:

1. Each procedure-name-1, procedure-name-3, etc., is the name of a paragraph containing a single sentence consisting of a GO TO statement without the DEPENDING option.

2. Each procedure-name-2, procedure-name-4, etc., is the name of a paragraph or section in the Procedure Division.

General Rules:

1. When the ALTER statement is executed, the GO TO statement in the paragraph named procedure-name-1, procedure-name-3, etc., is modified so that subsequent executions of the GO TO statement cause a transfer of control to the paragraphs or sections named procedure-name-2, procedure-name-4, etc., respectively.

2. A GO TO statement in a section whose priority-number is equal to or greater than 50 must not be referred to by an ALTER statement in a section having a different priority-number. All other uses of the ALTER statement are valid and are performed, even if the GO TO statement to which the ALTER statement refers is a fixed overlayable segment. Refer to Segmentation in Section II.

The CALL statement is used to transfer control to a separately compiled program or to an entry point within a program with a standard return mechanism provided.

General Format:

```
CALL { literal-1      } [ USING identifier-2
     { routine-name   }

     [, identifier-3 ]... ]
```

Syntax Rules:

1. Literal-1 must be a nonnumeric literal.

2. Routine-name designates a called program and may assume one of the following meanings:

   a. If the program being called is an independently compiled COBOL program, routine-name must be its PROGRAM-ID.

   b. If the routine-name being called is an explicit entry-name in an independently compiled COBOL program, the entry-name must be one specified in an ENTRY POINT phrase in the independent program.

   c. If the program being called has been developed via another language, routine-name must be the program's identification or entry-name according to the rules of that language.

   Series 60/6000 COBOL employs the CALL macro of GMAP which results in restrictions on the routine-name for a, b, or c above. Routine-name must not be more than six characters, at least one of which is nonnumeric and the first of which is nonzero.

3. USING identifiers must not be subscripted.

General Rules:

1. The state of the called program is undefined.

2. The implied entry point (PROGRAM-ID) of a program written in COBOL is the first nondeclarative procedural statement. This entry point is produced automatically by the compiler. The implied exit point follows immediately after the last procedural statement in the source program and is also produced automatically. It is the effective exit point when a program is called by its PROGRAM-ID program-name.

3. Any object program produced by the COBOL compiler can be called as a program from other object programs. COBOL object programs may also have multiple entry points which can be called and executed from other object programs. Normally, such programs should not contain a STOP RUN statement. The program or entry point procedures should be written to allow control to eventually pass to an appropriate exit point. An EXIT paragraph must be used if a program has alternative routes to the exit point. At least one EXIT entry-name must be specified for each ENTRY POINT phrase in a program.

4. A called program may CALL other COBOL programs by either explicit or implicit entry point. However, caution must be exercised to avoid a CALL into a program which has any previous active CALL still current. A program may not call its own PROGRAM-ID program-name or entry-names within itself.

5. The USING phrase is utilized to specify an argument list for the program being called. The USING arguments are not valid when a CALL references a COBOL program by its PROGRAM-ID program-name. They are valid when referencing explicit entry points within a program. The order and descriptions of the arguments must conform to the called program's requirements. At most, ten arguments may be specified.

6. The USING phrase specifies input and output arguments to the called program. USING identifiers must be defined working-storage data items with level-numbers of 01 or 77, or items in files for which a PROCESS AREA is specified. A USING argument may be a file-name, in which case the object program argument will be a File Control Block pointer. File-names are not valid arguments when calling a COBOL program; they are restricted to called programs developed in another language.

7. The number of USING arguments specified in a CALL to an ENTRY POINT must correspond exactly with the number of USING and GIVING arguments specified in its ENTRY POINT phrase and the data descriptions for each pair of corresponding arguments must be identical.

8. The CALL statement may appear anywhere within a segmented program. When a CALL statement appears in a section with a priority-number greater than or equal to 50, that segment is in its last used state when the EXIT PROGRAM statement in the called program returns control to the calling program.

The CLOSE statement is used to terminate the processing of reels, units, and files with optional rewind and/or lock where applicable.

General Format:

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

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

Syntax Rules:

1. Each file-name is the name of a file upon which the CLOSE statement is to operate; it must not be the name of a sort file or merge file.

2. The REEL and WITH NO REWIND options apply only to files stored on tape devices and other devices to which these terms are applicable. The▲ UNIT option is applicable only to mass storage files in the▲ sequential-access mode.

General Rules:

In the discussion below, the term 'unit' applies to all input-output devices; the term 'reel' applies to tape devices. Treatment of mass storage devices in the sequential-access mode is logically equivalent to the treatment of a file on tape or analogous media.

1. For the purpose of showing the effect of various CLOSE options as applied to various storage media, all input, output, and input-output files are divided into the following categories:

   a. Non-reel. A file whose input or output medium is such that the concepts of rewinding and reels have no meaning.

   b. Sequential single-reel/unit. A sequential file that is entirely contained on one unit.

   c. Sequential multi-reel. A sequential file that is contained on more than one reel.

   d. Random single-unit. A file in the random-access mode that is entirely contained on a single mass storage unit.

2. The results of executing each CLOSE option for each type of file are summarized in the following table; symbol definitions are given after the summary. If the definition depends on whether the file is an input or an output file, alternate definitions are given; otherwise, the one definition applies to input, output, and input-output files.

CLOSE Option	File Type			
	Non-Reel	Sequential Single-Reel/Unit	Sequential Multi-Reel	Random Single-Unit
CLOSE	C	C,G	C,G,A	C
CLOSE WITH LOCK	C,E	C,G,E	C,G,E,A	C,E
CLOSE WITH NO REWIND	X	C,B	C,B,A	X
CLOSE REEL	X	X	F,G	X
CLOSE REEL WITH LOCK	X	X	F,D	X
CLOSE REEL WITH NO REWIND	X	X	F,B	X

NOTE: The letters in the table are explained below.

A = Previous Reels Unaffected

Input files and input-output files: All reels in the file prior to the current reel are processed according to the standard reel swap procedure, except those reels controlled by a prior CLOSE REEL statement. If the current reel is not the last in the file, the reels in the file following the current one are not processed in any way.

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

B = No Rewind of Current Reel

The current reel is left in its current position.

C = Standard Close File

Input files and input-output files (sequential-access mode): If the file is positioned at its end and a label record is specified for the file, the label is processed according to the standard label convention. The behavior of the CLOSE statement when a label record is specified but not present, or when a label record is not specified but is present, is undefined. If specified by the USE statement, a user's label procedure is executed. The order of execution of these two processes is specified by the USE statement. If the file is positioned at its end and label records are not specified for the file, label processing does not take place but other closing operations are executed. If the file is positioned other than at its end, the closing operations are executed, but there is no ending label processing. An input file, or an input-output file, is considered to be at the end of the file if the imperative-statement in the AT END phrase of the READ statement has been executed and no CLOSE statement has been executed.

Input files and input-output files (random-access mode) and output files (random- or sequential-access mode): If a label record is specified for the file, the label is processed according to the standard label convention. The behavior of the CLOSE statement when a label record is specified but not present, or when a label record is not specified but is present, is undefined. If specified by the USE statement, a user's label procedure is executed. The order of execution of these two processes is specified by the USE statement. In addition, other closing operations are executed. If label records are not specified for the file, label processing does not take place but other closing operations are executed.

D = Standard Reel Lock

An appropriate technique is supplied to ensure that the current reel cannot be processed again as a part of this file during this execution of this object program. (The current reel is rewound.)

E = Standard File Lock

An appropriate technique is supplied to ensure that this file cannot be opened again during this execution of this object program.

F = Standard Close Reel

Input files: The following operations are executed:

- A reel swap.

- The standard beginning reel label procedure and the user's beginning reel label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.

The next executed READ statement makes available the next data record on the next reel.

Output files and input-output files: The following operations are executed:

- (For output files only.) The standard ending reel label procedure and the user's ending reel label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.
- A reel swap.
- The standard beginning reel label procedure and the user's beginning reel label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.

G = Rewind

The current reel or analogous device is positioned at the physical beginning of its content.

X = Illegal

X indicates an illegal combination of a CLOSE option and a file type. If any such combination is used, the results at object program execution are unpredictable.

3. All files that have been opened must be closed prior to the execution of a STOP RUN statement.

4. If the file has been specified with the OPTIONAL phrase in the FILE-CONTROL paragraph and is not present, the standard end-of-file processing is not performed.

5. If a CLOSE statement without the REEL option has been executed for a file, a READ, WRITE, or SEEK statement for that file must not be executed unless an intervening OPEN statement for that file is executed.

6. A CLOSE statement may not be executed as a part of a label procedure appearing in the declarative portion of a program.

The COMPUTE statement is used to assign the value of a numeric data item, a literal, or an arithmetic-expression to a data item.

General Format:

```
COMPUTE  identifier-1  [ROUNDED]

{ FROM   }        {identifier-2          }
{  =     }        {literal-1             }
{ EQUALS }        {arithmetic-expression }

    [ON  SIZE ERROR  imperative-statement]
```

Syntax Rules:

1. Literal-1 must be a numeric literal.

2. Each identifier must refer to an elementary numeric item. However, identifier-1 is not used as an operand and therefore may, if necessary, describe numeric data items containing editing symbols.

3. The maximum size of each operand is 18 decimal digits.

4. The arithmetic-expression, if used, permits any meaningful combination of identifiers, numeric literals, and arithmetic operators, parenthesized as required.

5. The words FROM and EQUALS are equivalent to each other and to the symbol '='. They may be used interchangeably; the choice is given for readability.

General Rules:

1. Identifier-1 is assigned the value computed from the arithmetic-expression, or the value of identifier-2, or of literal-1. The identifier-2 and literal-1 options provide alternative methods for setting the value of identifier-1 equal to the value of identifier-2 or literal-1.

2. The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, or DIVIDE.

3. Refer to the Common Options in Statement Formats paragraph for an explanation of the uses of the ROUNDED and SIZE ERROR options. For COMPUTE usage, refer to the COBOL User's Guide.

The COPY statement is used in the Procedure Division to incorporate paragraph procedures from a library into the source program.

Format 1:

```
{paragraph-name.                                          }
{section-name  SECTION  [priority-number] .               }

COPY  library-name

[ REPLACING   word-1   BY { word-2     }                  ]
[             {literal-1  }                               ]
[             {identifier-1}                              ]

[ , word-3   BY {word-4    }  ] ... ]  .
[               {literal-2 }        ]
[               {identifier-2}      ]
```

Format 2:

```
paragraph-name. . COPY  library-name  FROM  LIBRARY.
```

NOTE: Paragraph-name and section-name are not part of the COPY statement; they are shown only for clarity.

Syntax Rules:

1. When the COPY statement is specified, the library-name is required. The library-name must be identical to the name associated with the desired text on the library.
2. In Format 1, a word is any COBOL word and may be one of the following:
   - Condition-name
   - Data-name
   - File-name
   - Mnemonic-name
   - Procedure-name
3. When the COPY statement is used, it must be the first statement in the paragraph or section.

General Rules:

1. Format 1 of the COPY statement represents the American National Standard COPY function. **Format 2 represents the HIS COPY function.**

2. For a detailed description of the COPY statement, see Section VIII, the COBOL Library, and Section XIV of the COBOL User's Guide.

The DISPLAY statement is used to transmit low-volume data to a special output device.

Format 1:

```
DISPLAY { literal-1    }   [ , { literal-2    } ] ...
        { identifier-1 }       { identifier-2 }

    [UPON mnemonic-name]
```

Format 2:

```
DISPLAY { literal-1    }   [ , literal-2    ] ...   [ { END-OF-SEGMENT } ]
        { identifier-1 }       { identifier-2 }       { ESI           }
                                                      { END-OF-MESSAGE }
                                                      { EMI           }
                                                      { END-OF-TRANSACTION }
                                                      { ETI           }

    UPON mnemonic-name
```

Syntax Rules:

1. The mnemonic-name is associated with an I-O device in the SPECIAL-NAMES paragraph of the Environment Division.
2. Each literal may be any figurative constant except ALL.
3. The END-OF-MESSAGE phrase is assumed if no option is specified and the mnemonic-name is associated with COMMUNICATION-DEVICE.
4. The following abbreviations apply:
   a. ESI is an abbreviation for END-OF-SEGMENT indicator.
   b. EMI is an abbreviation for END-OF-MESSAGE indicator.
   c. ETI is an abbreviation for END-OF-TRANSACTION indicator.

General Rules:

Format 1:

1. A DISPLAY statement may be used to transmit output data to any of the following destinations:

   a. SYSOUT (the low-volume system output feature of the operating system).

   b. REMOTE (a terminal not operating under the control of the Transaction Processing System).

   c. CONSOLE and TYPEWRITER (the system operator interface).

   d. SWITCH (a portion of the program switch word, a special software feature provided by the operating system).

   When the UPON mnemonic-name phrase is not used, the output destination is considered to be system output (SYSOUT). The UPON phrase must be specified for any other output destination, and the mnemonic-name must be a user-supplied word associated with an output destination by a phrase in the SPECIAL-NAMES paragraph.

   A DISPLAY statement may not be executed, in an attempt to transmit data to system output (SYSOUT), as a part of a label procedure appearing in the declarative portion of a program.

2. The following rules apply to all DISPLAY statements:

   a. When the DISPLAY statement specifies multiple operands, the data characters associated with each operand are concatenated in the order of the occurrence of the operands. Operands are not automatically separated by spaces.

   b. The first character of the first operand is positioned in the first character position of a line, subject to the effects of any horizontal and vertical tabulation control characters embedded in the data.

   c. Identifier-1, identifier-2, ..., must be described with USAGE DISPLAY (explicitly or implicitly) or DISPLAY-1.

   d. Literal-1, literal-2, ..., may be figurative constants, in which case only a single occurrence of the figurative constant is displayed.

3. When utilizing the system output file (SYSOUT) via DISPLAY statements, the user may either omit the UPON phrase, or associate a mnemonic-name with SYSOUT in the SPECIAL-NAMES paragraph. This type of output is assumed to be directed to a printer, each line of which is considered to contain 132 character positions. The DISPLAY statement may produce more than one line of printing to SYSOUT if the cumulative size of the referenced operands exceeds a total of 132 characters.

4. The utilization of DISPLAY statements that transmit data to SYSOUT in a module overlay environment must be carefully planned to avoid possible overlay loading on top of the COBOL subroutine that controls output for SYSOUT displays. One method that may be used to avoid such an overlay is to place at least one DISPLAY statement in the main module that will never be overlayed. That statement need not actually be executed.

5. Each DISPLAY statement whose mnemonic-name is associated with REMOTE will cause from one to four lines to be displayed on the remote terminal, depending upon the size of the referenced data items. Each line will contain at most 72 characters, thereby limiting the total size of the referenced items to 288 characters.

6. Each DISPLAY statement whose mnemonic-name is associated with CONSOLE will allow from one to four lines to be displayed on the system console, depending on the size of the referenced data items. Each line will contain at most 72 characters, thereby limiting the total size of the referenced items to 288 characters. The output line (or, if more than one line results from the statement, the last output line) is held in a buffer until the next execution of an ACCEPT statement associated with CONSOLE. At that time, the line will be used to inform the operator of a pending need for a response.

   The ACCEPT and DISPLAY statements need not appear together in the source program, provided that the DISPLAY statement is executed first. Should no ACCEPT statement be executed after the DISPLAY statement, the output data is not displayed. Two DISPLAY statements of this kind with no intervening ACCEPT statement would result in the suppression of the output from the first DISPLAY statement. If more than one line results from an execution of a DISPLAY statement associated with CONSOLE, all lines except the last line are emitted at once and, in a multiprogramming environment, there is no assurance that any of the lines will be juxtaposed on the console display.

7. Each DISPLAY statement whose mnemonic-name is associated with TYPEWRITER will cause from one to four lines to be displayed on the system console, depending on the size of the referenced data items. Each line will contain at most 72 characters, thereby limiting the total size of the referenced items to 288 characters. If more than one line is emitted for a given DISPLAY statement, there is no assurance, in a multiprogramming environment, that the lines will be juxtaposed on the console display. System console output is not recommended unless very unusual circumstances exist.

8. If a mnemonic-name associated with SWITCH is specified, only one operand (data-name, literal, or the figurative constant ZERO) may be given. If the value of the operand is 1, the switch will be set ON; if the value is 0, the switch will be set OFF. If a literal is used, it must be an integer that has a value of 1 or 0. If a data-name is specified, it must be a **COMPUTATIONAL-1** data item in the Working-Storage Section, with a size not exceeding eight digits. The following data description is recommended:

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

   If the value of the item exceeds one (1), the value modulo 2 determines the switch setting.

9. Refer to Section VI of the COBOL User's Guide for additional information.

Format 2:

1. In Format 2, mnemonic-name must be associated with COMMUNICATION-DEVICE in the SPECIAL-NAMES paragraph. The DISPLAY statement is used to deliver a message via the transaction processing intercom facility.

2. Output to a terminal must not exceed the physical limits of the terminal.

3. When a message is to be delivered via the intercom facility, the destination ID in the output header (DESTINATION) is used as the logical terminal ID by the transaction processing output interface routine. DEST-COUNT will be set to one (1) and DESTINATION (1) will be initialized to the input identifier (SOURCE-ID) if the Transaction Processing Applications Program (TPAP) does not initialize those fields prior to the execution of each DISPLAY statement.

4. The output transaction number in the output header must be initialized by the TPAP prior to the execution of each DISPLAY statement if it differs from the input transaction number in the input header.

5. The END-OF-SEGMENT (ESI), END-OF-MESSAGE (EMI), and END-OF-TRANSACTION (ETI) options can be used only in a DISPLAY statement that references a mnemonic-name associated with COMMUNICATION-DEVICE.

   a. END-OF-SEGMENT (ESI) indicates that the operand(s) of the DISPLAY statement constitute(s) a message segment. More segments may follow to complete the message.

   b. END-OF-MESSAGE (EMI) indicates that this message is the end of a message for the specified destination. More messages may follow using the same transaction number.

   c. END-OF-TRANSACTION (ETI) indicates that this message text is the end of the last message for the specified transaction number. It is necessary to provide a DISPLAY of END-OF-TRANSACTION (ETI) in order to continue processing another transaction.

6. Format 2 of the DISPLAY statement may also be used to invoke another TPAP program. For information concerning the transaction processing feature, refer to Section VII of the COBOL User's Guide and to the Transaction Processing System User's Guide.

The DIVIDE statement is used to divide one numeric data item into another and to set the value of a data item equal to the result.

Format 1:

```
DIVIDE {identifier-1} INTO identifier-2 [ROUNDED]
       {literal-1   }

    [, identifier-3 [ROUNDED] ] ...

    [ON SIZE ERROR imperative-statement]
```

Format 2:

```
DIVIDE {identifier-1} INTO {identifier-2} GIVING
       {literal-1   }      {literal-2   }

    identifier-3 [ROUNDED]  [, identifier-4 [ROUNDED] ] ...

    [ON SIZE ERROR imperative-statement]
```

Format 3:

```
DIVIDE {identifier-1} BY {identifier-2} GIVING
       {literal-1   }    {literal-2   }

    identifier-3 [ROUNDED]  [, identifier-4 [ROUNDED] ] ...

    [ON SIZE ERROR imperative-statement]
```

Format 4:

```
DIVIDE {identifier-1} INTO {identifier-2} GIVING
       {literal-1   }      {literal-2   }

        identifier-3 [ROUNDED] REMAINDER identifier-4

       [ON SIZE ERROR imperative-statement]
```

Format 5:

```
DIVIDE {identifier-1} BY {identifier-2} GIVING
       {literal-1   }    {literal-2   }

        identifier-3 [ROUNDED] REMAINDER identifier-4

       [ON SIZE ERROR imperative-statement]
```

Format 6:

```
DIVIDE identifier-1 BY identifier-2
```

Syntax Rules:

1. Each identifier must refer to an elementary numeric item, except that any identifier associated with the GIVING or REMAINDER phrase must refer to either an elementary numeric item or an elementary numeric edited item.

2. Each literal must be a numeric literal.

3. The maximum size of each operand is 18 decimal digits. The composite of operands (the hypothetical data item resulting from the superimposition of all operands of a given statement, excluding the REMAINDER data item, aligned on their decimal points) must not contain more than eighteen digits.

General Rules:

1. When Format 1 is used, the value of identifier-1 or literal-1 is divided into the value of identifier-2. The value of the dividend (identifier-2) is replaced by the quotient. This process also occurs for identifier-1 or literal-1 and identifier-3, etc.

2. When Format 2 is used, the value of identifier-1 or literal-1 is divided into identifier-2 or literal-2 and the quotient is stored in identifier-3, identifier-4, etc.

3. When Format 3 is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2 and the quotient is stored in identifier-3, identifier-4, etc.

4. Formats 4 and 5 are used when a remainder from the division operation is desired, namely identifier-4. The remainder in COBOL is defined as the result of subtracting the product of the quotient (identifier-3) and the divisor from the dividend. If ROUNDED is used, the quotient used to calculate the remainder is an intermediate field which contains the quotient of the DIVIDE statement, truncated rather than rounded.

5. In Formats 4 and 5, the accuracy of the REMAINDER data item (identifier-4) is defined by the calculation described above. Appropriate decimal alignment and truncation (not rounding) will be performed for the contents of the data item referenced by identifier-4, as needed.

6. When the ON SIZE ERROR phrase is used in Formats 4 and 5, and a size error occurs on the quotient, the contents of identifier-3 will not be changed but the result of the divide will be used in computing the remainder.

   If the size error occurs on the remainder, the contents of the data item referenced by identifier-4 will remain unchanged. However, as with other instances of multiple results of arithmetic statements, analysis must be performed to determine which situation has occurred.

7. In Format 6, the initial value of the dividend (identifier-1) will be replaced by the computed value of the quotient obtained from the division. This nonstandard format is not recommended.

8. Refer to the Common Options in Statement Formats paragraph in this section for uses of the ROUNDED and SIZE ERROR options and multiple results.

The ENTER statement is used to conserve time or space in the execution of the object program or to include specific 'statements' in the source program that are not defined in the COBOL language.

Format 1:

```
ENTER { TIME-SAVING  } MODE.
      { SPACE-SAVING }
```

Format 2:

```
[paragraph-name.]      ENTER GMAP.
                         .
                         .
                         .      } GMAP coding
                         .
                         .
                       ENTER COBOL.
```

Format 3:

```
ENTER DEFINITIONS.

 IDS SIZE integer EQUALS data-name.

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

ENTER COBOL.
```

Format 4:

```
ENTER LINKAGE MODE.
    {  CALL routine-name [ USING data-name-1 [, data-name-2] ... ]
    {  ENTRY POINT IS entry-name
    {       [ [ USING data-name-1 [, data-name-2] ... ]
    {         [ GIVING data-name-3 [, data-name-4] ... ] ]
    {
    POPUP procedure-name
```

ENTER COBOL.

Syntax Rules:

1. In Format 3, integer must be a six-digit numeric literal.
2. In Format 3, literal-1 must be a nonnumeric literal, consisting of one to six characters which satisfy the GMAP rules for symbol formation.
3. In Format 3, data-name must be defined in the Working-Storage Section as a record description entry and may be qualified but not subscripted. Procedure-name must be defined in the Procedure Division as a paragraph-name or section-name and may be qualified (if it is a paragraph-name). COBOL prohibits use of the same name for both data and procedures.

General Rules:

1. ENTER statements in a source program written for a computer other than the one on which the current source program is being compiled must be changed.

Format 1:

1. Under some circumstances, the compiler is capable of producing two alternative sets of object program instructions for a given procedural statement. The TIME-SAVING version requires more space but less execution time than the SPACE-SAVING version. (Generally, such alternatives are available in the compiler only when the tradeoff exceeds two-for-one in both categories.) This ENTER statement permits the user to take advantage of such possible alternatives considering the relative level of activity of various parts of the program.

2. The normal mode is TIME-SAVING. The SPACE-SAVING mode applies to all statements following ENTER SPACE-SAVING until ENTER TIME-SAVING is encountered, at which point the compiler resumes the normal TIME-SAVING mode.

3. For certain procedural statements, the TIME-SAVING or SPACE-SAVING MODE options are not necessarily significant. In such procedures, the mode entered has no effect on the object program.

Format 2:

1. The ENTER GMAP statement must not contain a comment; a comment will cause GMAP flags to be appended to the generated code. The GMAP coding following an ENTER GMAP statement must be terminated by an ENTER COBOL statement which must begin on a new line. The ENTER COBOL statement must not contain a paragraph-name. The lines between the ENTER GMAP and ENTER COBOL statements must consist of GMAP coding.

2. The first GMAP line following the ENTER GMAP statement should be either an executable GMAP instruction or a NULL pseudo-operation. If the COBOL statement immediately preceding the ENTER GMAP statement implies a 'next sentence', a symbol will be generated and assigned to the first GMAP line. If the first GMAP line is a line for which a symbol is not allowed, the generated symbol will be undefined. If it is inconvenient to ensure that the first GMAP line is an executable instruction, the user should:

   a. Place a paragraph-name immediately preceding the ENTER GMAP statement, or

   b. Ensure that the COBOL statement immediately preceding the ENTER GMAP statement does not imply a 'next sentence', or

   c. Follow the ENTER GMAP statement with a NULL pseudo-operation.

3. The special format used for the GMAP coding is the standard GMAP format shifted six places to the right.

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

Information appearing to the right of column 72 is not interpreted as part of the variable field. As in ordinary GMAP coding, comments must be separated from variable field information by at least one blank. The quotation mark character (") should not be used in the variable field because it has a special meaning in COBOL; the word QUOTE should be used instead.

4. GMAP symbols defined in the location field must not conflict with reserved system symbols. They must follow the GMAP rules for symbol formation. Symbols reserved for compiler use which must not be defined in the location field of GMAP statements include:

   a. Symbols in the form lnnnnn, where 'l' is any letter and 'nnnnn' is any string of five digits.

   b. Symbols having any file-code specified in the Environment Division as their leftmost two characters.

   c. Symbols of the form VfcEOF, where 'fc' is any file-code specified in the Environment Division.

   d. The PROGRAM-ID program-name defined in the Identification Division.

   e. The symbol ENTER.

   f. Symbols having either '.C' or 'C.' as their leftmost two characters.

   g. The special symbols ZINBUF, ZOTHDR, ZOUTB1, ZOUTB2, COMMI, COMMO.

   h. Any symbol starting in column 7 with the first character numeric (such a symbol is treated as an optional debug statement).

5. COBOL data-names and procedure-names are not directly accessible to GMAP coding; a special provision (see Format 3) permits GMAP symbols to be applied to COBOL procedures and data items. The 'locsym' of each File Control Block is the two-character file-code assigned in a SELECT sentence in the Environment Division followed by the characters 'FICB'. (Refer to the File and Record Control reference manual.)

   For a file with an explicit or implicit process area, the beginning location of the process area has the symbol fcRECD where 'fc' is the file-code assigned. For any other file, a current record location pointer is available (provided the file is OPEN) in the address field (bits 0-17) of the File Control Block word with symbol fcFICB, where 'fc' is the file-code assigned. Data must be addressed relative to the current record location pointer for such a file.

   References to File Control Blocks make a program sensitive to changes in input-output software and are therefore strongly deprecated.

6. All GMAP rules must be observed in the GMAP coding. Pseudo-operations that alter the location counter may cause unpredictable results at object program execution. Their use presupposes a thorough understanding of the location counter conventions followed by the COBOL compiler in generating coding.

7. The user must ensure that the contents of the EIS address registers are intact or reinitialized after every ENTER GMAP/ENTER COBOL sequence. The compiler implicitly optimizes the use of the EIS address registers across paragraphs. A given register will not be reinitialized as long as the compiler can determine from the source statements that the register data is intact. Coding following the ENTER GMAP statement is not analyzed by the compiler; thus, when coding following the ENTER GMAP statement is used either to CALL a subroutine or to modify code generated from COBOL source statements, the user must either protect all index and address registers having GMAP coding or indicate to the compiler that the registers are to be reinitialized. This is accomplished by immediately following the ENTER COBOL statement with a new paragraph and including at least one 'ENTRY POINT IS entry-name' phrase in the program.

Format 3:

1. The ENTER DEFINITIONS statement allows GMAP location symbols to be associated with COBOL data-names (or their properties) and procedure-names.

2. The IDS SIZE phrase allows I-D-S/COBOL programs to specify the size of a data-name for internal checking purposes. It has no effect other than to provide a warning message on the listing when such a check fails. Normally, it is a statement produced by the I-D-S translator. Integer must be given as exactly six characters.

3. A SYMBOL phrase that references a data-name using the SIZE option causes the compiler to equate the symbol to an appropriate number that indicates the number of BCD character positions required to contain the data-name.

4. A SYMBOL phrase that references a data-name using the INITIAL option causes the compiler to equate the symbol to an appropriate number (0 through 5 for BCD fields or either 0 through 7 or 4 through 7 for packed decimal fields) indicating the first character position occupied by the data item within the (first) computer word. For BCD fields, the leftmost position is indicated by 0; the rightmost by 5. For packed decimal fields, the leftmost position is indicated by 0 or 4 and the rightmost position is indicated by 3 or 7, depending on whether the data item begins on the word or half-word boundary.

5. A SYMBOL phrase that references a data-name or a procedure-name only causes the compiler to equate the given GMAP symbol (literal-1) to the first memory location of the specified data area or procedural instructions. A level 01 data area described in the File Section may not be equated to a symbol (literal-1) unless a process area will be present for the file and the compiler symbol (fcRECD) is used instead of the source data-name.

Format 4:

1. CALL permits transfer of control to a separately compiled program or entry point within a program with a standard return mechanism provided. (The generated coding employs the GMAP CALL pseudo-operation.) CALL may also be used without the ENTER LINKAGE statement. (Refer to the CALL statement.) The following conventions govern the use of CALL:

   a. If the program being called is an independently compiled COBOL program, routine-name must be its PROGRAM-ID.

   b. If the routine-name being called is an explicit entry-name in an independently compiled COBOL program, the entry-name must be one specified in an ENTRY POINT phrase in the independent program.

   c. If the program being called has been developed via another language, routine-name must be the program's identification or entry-name according to the rules of that language.

   d. The implied entry point (PROGRAM-ID) of a program written in COBOL is the first nondeclarative procedural statement. This entry point is produced automatically by the compiler. The implied exit point follows immediately after the last procedural statement in the source program and is also produced automatically. It is the effective exit point when a program is called by its PROGRAM-ID program-name.

   e. Any object program produced by the COBOL compiler can be called as a program from other object programs. COBOL object programs may also have multiple entry points that can be called and executed from other object programs. Normally, such programs should not contain STOP RUN. The program or entry point procedures should be written to allow control to eventually pass to an appropriate exit point. An EXIT paragraph must be used if a program has alternative routes to the exit point. At least one EXIT entry-name must be specified for each explicit entry-name in a program.

   f. A called program may CALL other COBOL programs by implied entry point or by entry-names specified within the other programs. Entry point procedures may CALL entry-names in other programs; however, caution must be exercised to avoid a CALL into a program which has any previous active CALL still current. A program must not call its own PROGRAM-ID program-name or entry-names within itself.

   g. The USING phrase is utilized to specify an argument list for the program being called. The USING arguments are not valid when a CALL references a COBOL program by its PROGRAM-ID program-name. They are valid when referencing explicit entry points within a program. The order and descriptions of the arguments must conform to the called program's requirements. At most, ten arguments may be specified.

h. The USING phrase specifies 'input' and 'output' arguments to the called program. USING data-names must reference working-storage data items with level-numbers of 01 or 77, or items in files for which a PROCESS AREA is specified. A USING argument may be a file-name, in which case the object program argument will be a file control table pointer. File-names are not valid arguments when calling a COBOL program; they are restricted to called programs developed in another language.

i. USING data-names must not be subscripted.

j. The number of USING arguments specified in a CALL to an explicit entry-name must correspond exactly with the number of USING and GIVING arguments specified in its ENTRY POINT phrase and the data descriptions for each corresponding argument must be identical. Arguments may not be larger than 256 words.

2. The ENTRY POINT phrase provides a way to define entry points into a program other than the implied entry point (the PROGRAM-ID program-name) which is the first nondeclarative procedural statement.

The following conventions govern the use of ENTRY POINT:

a. The entry-name specified must not contain more than six characters, at least one of which is nonnumeric and the first of which is nonzero. The entry-name must not be the same as the PROGRAM-ID program-name. Refer to the reserved GMAP location symbols in Appendix F of the COBOL User's Guide.

b. An ENTRY POINT phrase can be used in any location in the Procedure Division except within the declarative portion.

c. There must be no path of program flow to an ENTRY POINT phrase within the program containing the ENTRY POINT phrase.

d. An ENTRY POINT phrase may only be referenced by calls from external programs. It must not be referenced by a CALL from within the program in which it resides.

e. The user may specify as many ENTRY POINT phrases as desired for a particular subprogram.

f. For each ENTRY POINT phrase there must be at least one EXIT entry-name statement to provide the exit point for any CALL which references the entry-name.

g. The USING phrase specifies an input argument list for an ENTRY POINT phrase. The USING data-names must be level 77 or 01 items specified in the Working-Storage Section. When a CALL references an entry point with USING arguments, the compiler generates word moves for each argument, using the indirect address specified in the CALL as a sending field and the corresponding data-name from the USING argument list of the entry point as a receiving field. The descriptions of the corresponding arguments must be identical. Procedural statements following a particular entry point will not be executed until all USING argument moves are complete.

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

   The GIVING option uses the corresponding indirect address specified in the CALL as a receiving field and the corresponding data-name from the GIVING argument list for the entry point as a sending field.

i. The user must assure a one-to-one correspondence between the CALL USING arguments and the ENTRY POINT USING/GIVING arguments. The data formats for the corresponding arguments must be identical. Arguments may not be larger than 256 words.

j. USING/GIVING data-names must not be subscripted.

k. An entry-name must not be used on a $ ENTRY card.

3. The POPUP option permits extended use of PERFORM statements. Each executed PERFORM statement causes an exit link to enter a pushdown stack which services all PERFORMs. The stack is arranged so that only the exit mechanism for the latest PERFORM executed may be engaged. If control passes to any other exit mechanism, even if a relevant PERFORM is active, control is passed to the next statement in the written sequence. When the exit mechanism of the latest executed PERFORM is reached, the exit link is removed and the stack is 'popped up' before control is returned. This stack structure permits recursive performs, nested performs with crossing ranges, and multiple active performs with a common exit point. (If such extended capabilities are employed, the user must supply moves as necessary to avoid unintended overlaying of data with new values.) The pushdown stack approach inherently requires that all 'popups' eventually occur in an orderly manner. This means that control must eventually reach the exit mechanism of any active PERFORM; and for nested performs control must pass to the exit mechanisms in the logical order 'innermost to outermost'. POPUP permits this inherent control to be circumvented. When the control sequence is prepared in such a way that not all exit mechanisms will be operated in the required order, POPUP must be employed to remove the unused exit links from the stack. The simplest example of such a situation is a GO TO statement transferring control outside of a PERFORM range without a subsequent return to the range.

The following rules apply to POPUP:

a. Procedure-name must be the name of the last paragraph or section in the range of a currently active PERFORM.

b. POPUP removes all exit links from the stack up to and including those associated with the exit mechanism following the 'procedure-name' paragraph or section. POPUP does not return control for any exit link; it merely passes control to the statement following it in the source program after its action on the stack.

c. If control permanently leaves a PERFORM range, the memory area allocated for the stack will eventually overflow if the PERFORM is repeatedly executed unless POPUP is used. Stack overflow aborts the object program.

d. If the exit mechanism of an active PERFORM is deliberately bypassed, the exit mechanisms for prior PERFORM statements which are still active cannot return control unless POPUP is used.

e. POPUP must not reference an entry-name specified by the ENTRY POINT phrase.

The EXAMINE statement is used to replace and/or count the number of occurrences of a given character in a data item.

Format 1:

```
EXAMINE identifier TALLYING  { UNTIL FIRST }  literal-1
                             { ALL         }
                             { LEADING     }

    [REPLACING BY  literal-2]
```

Format 2:

```
EXAMINE identifier REPLACING  { ALL                }  literal-3
                              { LEADING            }
                              { [UNTIL ] FIRST     }

      BY  literal-4
```

Syntax Rules:

1. The description of 'identifier' must be such that its usage is DISPLAY (explicitly or implicitly).
2. Each literal must consist of a single character belonging to a class consistent with that of 'identifier'. In addition, each literal may be any figurative constant except ALL.
3. A signed numeric literal is not permitted in the EXAMINE statement.

General Rules:

1. Examination proceeds as follows:
   a. For nonnumeric data items, examination starts at the leftmost character and proceeds to the right. Each character in the data item specified by the identifier is examined in turn.
   b. If a data item referred to by the EXAMINE statement is numeric, it must consist of numeric characters and may possess an operational sign. Examination starts at the leftmost character and proceeds to the right. Each character is examined in turn. If the letter 'S' is used in the PICTURE character-string of the data item description to indicate the presence of an operational sign, the sign is completely ignored by the EXAMINE statement.

2. The TALLYING phrase creates an integral count which replaces the value of a special register called TALLY. The count represents the number of:

   a. Occurrences of literal-1 when the ALL option is used.

   b. Occurrences of literal-1 prior to encountering a character other than literal-1 when the LEADING option is used.

   c. Characters not equal to literal-1 encountered before the first occurrence of literal-1 when the UNTIL FIRST option is used.

3. When any of the options in the REPLACING phrase are used, the replacement rules are as follows, subject to the qualifications stated in General Rule 2:

   a. When the ALL option is used, then literal-2 or literal-4 is substituted for each occurrence of literal-1 or literal-3.

   b. When the LEADING option is used, the substitution of literal-2 or literal-4 terminates as soon as a character other than literal-1 or literal-3 or the right-hand boundary of the data item is encountered.

   c. When the UNTIL FIRST option is used, the substitution of literal-2 or literal-4 terminates as soon as literal-1 or literal-3 or the right-hand boundary of the data item is encountered.

   d. When the FIRST option is used, the first occurrence of literal-1 or literal-3 is replaced by literal-2 or literal-4.

The EXIT statement is used to define the exit point for a series of procedures or to define the exit point for the logical end of a called program.

General Format:

paragraph-name.

```
EXIT [ { PROGRAM    } ]
       { entry-name }
```

Syntax Rules:

1. The EXIT statement must appear in a sentence by itself.

2. The EXIT sentence must be preceded by a paragraph-name and must be the only sentence in the paragraph.

General Rules:

1. It is sometimes necessary to transfer control to the end point of a series of procedures. This is usually accomplished by transferring control to the next paragraph or section, but in some cases this does not have the required effect. (For example, the point to which control is to be transferred may be at the end of a range of procedures governed by a PERFORM, a SORT, a MERGE, or could be at the end of a declarative section.) The EXIT statement is provided to enable a procedure-name to be associated with such a point. If such a procedure has only a single control path to the exit point, the EXIT statement is not required. If the procedure has alternative paths to the exit point, an EXIT paragraph should be employed, and the various paths should then transfer control to the EXIT paragraph.

2. The PERFORM statement, the SORT statement's INPUT PROCEDURE and OUTPUT PROCEDURE phrases, and the MERGE statement's OUTPUT PROCEDURE phrase require procedure-names referenced as follows:

   ```
   ...procedure-name-1 [ THRU procedure-name-2 ]
   ```

   The end of procedure-name-2 is the 'exit point' if the THRU option is used; otherwise the end of procedure-name-1 is the exit point. The presence of intervening EXIT paragraphs does not affect this rule. If an EXIT paragraph is needed, it must be placed at the proper exit point.

3. If control reaches an EXIT statement without the PROGRAM or entry-name option and no associated PERFORM, SORT, MERGE, or USE statement is active or if control reaches an EXIT PROGRAM statement and no CALL statement is active, control falls through the EXIT point to the first sentence of the next paragraph.

4. If control reaches an EXIT PROGRAM statement while the program is being executed as the result of a CALL statement, control returns to the point in the calling program immediately following the CALL statement.

5. The EXIT PROGRAM or EXIT entry-name statement must not be used in the declarative section.

6. The EXIT entry-name option defines an exit point for a specific entry-name which is given in the associated ENTRY POINT phrase. The entry-name must be defined by the ENTRY POINT phrase in the same program as the EXIT entry-name.

7. More than one exit point may be specified for any given entry point, but there must be at least one EXIT entry-name for each entry point specified.

8. If control reaches an EXIT entry-name statement, a check is made to determine if the entry-name corresponds to the currently active entry point. If they correspond, the exit mechanism passes the GIVING arguments specified for the entry point, restores all registers, and returns control to the point immediately following the CALL statement for the given entry-name. If the EXIT does not correspond to the entry-name, control will pass through the EXIT entry-name statement to the first sentence of the next paragraph.

The GENERATE statement is used to present a report entry based on Procedure Division control.

General Format:

GENERATE identifier

Syntax Rule:

1. Identifier represents a TYPE DETAIL report group or an RD entry.

General Rules:

1. If identifier represents the name of a TYPE DETAIL report group, GENERATE performs all the relevant automatic operations and produces an actual output DETAIL report group (detail reporting).

2. If identifier is the name of an RD entry, GENERATE performs all the relevant automatic operations and updates the FOOTING report group(s) within the report without producing an actual DETAIL report group associated with the report. Thus it increments all SUM counters associated with the report description (summary reporting). If the report includes more than one TYPE DETAIL report group, all SUM counters are incremented each time such a GENERATE is executed.

3. GENERATE produces the following automatic operations (as needed):

   a. Steps and tests the LINE-COUNTER to produce appropriate PAGE or OVERFLOW FOOTING and/or PAGE or OVERFLOW HEADING report groups. Increments PAGE-COUNTER when a PAGE or OVERFLOW condition is determined.

   b. Recognizes any specified CONTROL breaks to produce appropriate CONTROL FOOTING and/or CONTROL HEADING report groups.

   c. Accumulates all specified identifiers into the SUM counters. Resets the SUM counters on an associated control break. Performs an updating procedure between control break levels for each set of SUM counters.

   d. Executes any specified routines defined by the BEFORE REPORTING phrase of a USE statement before producing the associated report group(s).

4. During the execution of the first GENERATE statement referring to a report or to a DETAIL report group within a report, all CONTROL HEADING report groups specified for the report are produced in the order major...minor, immediately followed by any DETAIL report group specified in the statement. If an identifier control break is recognized when a GENERATE statement is executed (other than the first one executed for a report), all CONTROL FOOTING report groups specified for the report are produced from the minor report group up to and including the report group specified for the identifier which caused the control break. The CONTROL HEADING report group(s) specified for the report, from the report group specified for the identifier which caused the control break down to the minor report group, are then produced in that order. The DETAIL report group specified in the GENERATE statement is then produced.

5. When data is implicitly moved to a report group description entry, it is edited according to the rules described under the MOVE statement.

The GO TO statement is used to transfer control from one part of the Procedure Division to another.

Format 1:

GO TO [procedure-name-1]

Format 2:

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

, procedure-name-n DEPENDING ON identifier

Syntax Rules:

1. Each procedure-name must be the name of a paragraph or section in the Procedure Division.
2. Identifier is the name of a numeric elementary item described with no positions to the right of the assumed decimal point.
3. A GO TO...DEPENDING ON statement requires more than one procedure-name.

General Rules:

1. When a Format 1 GO TO statement is executed, control is transferred to procedure-name-1 or to another procedure-name if the GO TO statement was modified by an ALTER statement.
2. If procedure-name-1 in Format 1 is not specified, an ALTER statement that refers to this GO TO statement must be executed prior to the execution of this GO TO statement.
3. When the Format 1 GO TO statement is referred to by an ALTER statement, the following rules apply whether or not procedure-name-1 is specified:
   a. The GO TO statement must have an associated paragraph-name.
   b. The GO TO statement must be the only statement in the paragraph.

4. When a Format 2 GO TO statement is executed, control is transferred to procedure-name-1, procedure-name-2,..., procedure-name-n, depending on the value of the identifier being 1, 2, ..., n. If the value of identifier is anything other than the positive or unsigned integers 1, 2, ..., n, then a transfer does not occur and control passes to the next statement in the normal sequence for execution.

An IF statement causes a condition to be evaluated. The subsequent action of the object program depends on whether the value of the condition is true or false.

General Format:

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

Syntax Rules:

1. Statement-1 and statement-2 represent either a conditional statement or an imperative-statement, and either may be followed by a conditional statement.

2. The 'ELSE NEXT SENTENCE' phrase may be omitted only if it immediately precedes the final period of the sentence.

3. 'Conditions' are described at the beginning of this section.

General Rules:

1. When an IF statement is executed, the following action takes place:

   a. If the condition is true, the statements immediately following the condition (represented by statement-1) are executed and control then passes implicitly to the next sentence.

   b. If the condition is false, either the statements following ELSE are executed or, if the ELSE phrase is omitted, the next sentence is executed.

2. When an IF statement is executed and the NEXT SENTENCE phrase is present, control passes explicitly to the next sentence depending on the truth value of the condition and the placement of the NEXT SENTENCE phrase in the statement.

3. Statement-1 and statement-2 may contain an IF statement. In this case, the IF statement is said to be nested.

   IF statements within IF statements may be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered is considered to apply to the immediately preceding IF that has not been already paired with an ELSE.

4. When control is transferred to the next sentence, either implicitly or explicitly, control passes to the next sentence as written or to a return mechanism of a PERFORM or a USE statement.

The INITIATE statement is used to begin processing a report.

General Format:

```
INITIATE { report-name-1 , report-name-2 ... }
         { ALL                               }
```

Syntax Rule:

1. Each report-name specified in an INITIATE statement must be defined by a report description entry in the Data Division.

General Rules:

1. INITIATE resets to zero all data-name entries associated with this report which contain SUM clauses (SUM counters).
2. The PAGE-COUNTER, if specified, is initially set to one (1). After the execution of an INITIATE statement for a given report, the contents of PAGE-COUNTER for that report may be changed using a Procedure Division statement.
3. The LINE-COUNTER, if specified, is automatically set to zero (0) by the INITIATE statement.
4. A report can be reinitiated after it has been terminated.
5. If the ALL phrase is specified, all reports defined in the Report Section of the Data Division are initiated.
6. INITIATE does not open the file with which the report is associated. An OPEN statement for the file must be executed before the INITIATE statement.

The MERGE statement combines two or more identically sequenced files on a set of specified keys, and during the process makes records available, in merged order, to an output procedure or to an output file.

General Format:

```
MERGE file-name-1  ON { DESCENDING } KEY data-name-1  [, data-name-2 ] ...   *
                      { ASCENDING  }

          [ ON { DESCENDING } KEY data-name-3                                 *
               { ASCENDING  }

            [, data-name-4 ] ... ] ...

  USING file-name-2, file-name-3 [, file-name-4 ]...

{ OUTPUT PROCEDURE IS section-name-1 [ { THRU    } section-name-2 ] }
{ GIVING file-name-5                  { THROUGH }                   }
```

Syntax Rules:

1. File-name-1 must be described in a sort-merge file description entry in the Data Division.
2. Section-name-1 represents the name of an output procedure.
3. File-name-2, file-name-3, file-name-4, and file-name-5 must be described in a file description entry, not in a sort-merge file description entry, in the Data Division. The actual size of the logical record(s) described for file-name-2, file-name-3, file-name-4, and file-name-5 must be equal to the actual size of the logical record(s) described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, the corresponding records must be described in such a manner to cause equal amounts of memory to be allocated for the corresponding records.
4. The data-names may be qualified.
5. The words THRU and THROUGH are equivalent.

6. No more than one file-name from a multiple file reel can appear in the MERGE statement.

7. File-names must not be repeated within the MERGE statement.

8. MERGE statements may appear anywhere except in the declarative portion of the Procedure Division or in an input or output procedure associated with a SORT or MERGE statement.

General Rules:

1. The data-names following the word KEY are listed from left to right in the MERGE statement in order of decreasing significance without regard to how they are divided into KEY phrases. In the format, data-name-1 is the major key, data-name-2 is the next most significant key, etc.

   a. When the ASCENDING phrase is used, the merged sequence will be from the lowest value of the contents of the data items identified by the KEY data-names to the highest value, according to the rules for comparison of operands in a relation condition.

   b. When the DESCENDING phrase is used, the merged sequence will be from the highest value of the contents of the data items identified by the KEY data-names to the lowest value, according to the rules for comparison of operands in a relation condition.

2. When more than one record description entry appears in a merge file description, the KEY data items need be described in only one of the record description entries. Each KEY data item must occur in every data record of the merge file and must have the same relative position and actual format in all records. The PICTURE and USAGE of a given KEY data item must be the same for all records on the merge file. If a KEY data item is synchronized or justified, it must be identically synchronized or justified in all records on the merge file. The KEY data item descriptions must not contain an OCCURS clause nor be subordinate to entries containing an OCCURS clause. The KEY data items must not require subscripting or indexing. The dominant record size is set equal to the size of the first record described in the sort-merge file description entry (SD) in the Data Division. (See Section IX, File Ordering - SORT and MERGE, in the COBOL User's Guide.)

3. The MERGE statement will merge all records contained on file-name-2, file-name-3, and file-name-4. The files referenced in the MERGE statement must not be open when the MERGE statement is executed. These files are automatically opened and closed by the merge process with all implicit functions performed, such as the execution of any associated USE procedures. The terminating function for all files is performed as if the CLOSE statement had been explicitly written without optional phrases.

4. The output procedure, if present, must consist of one or more sections that are written consecutively in the source program and do not form a part of any other procedure. The output procedure must include at least one RETURN statement in order to make merged records available for processing. Control must not be passed to the output procedure except when a related SORT or MERGE statement is being executed. The output procedure may consist of any procedures needed to select, modify, or copy the records that are being returned, one at a time in merged order, from file-name-1. The restrictions on the procedural statements within the output procedure are:

   a. The output procedure must contain no transfers of control to points outside the output procedure; ALTER, GO TO, and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedure. COBOL statements that cause an implied transfer of control by the compiler to the declarative section are allowed.

   b. The output procedure must contain no SORT or MERGE statements.

   c. The remainder of the Procedure Division must contain no transfers of control to points inside the output procedures; ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division are not permitted to refer to procedure-names within the output procedure.

5. If OUTPUT PROCEDURE is specified, control passes to it during execution of the MERGE statement. The compiler inserts a return mechanism at the end of the last section in the output procedure. When control passes from the last statement in the output procedure, the return mechanism provides for termination of the merge procedure, and then passes control to the next statement after the MERGE statement. Before entering the output procedure, the merge procedure reaches a point at which it can select the next record in merged order when requested. The RETURN statements in the output procedure are the requests for the next record.

6. Segmentation, as defined in the COBOL User's Guide, can be applied to programs containing the MERGE statement. However, the following restrictions apply:

   a. If the MERGE statement appears in a section that is not in an independent segment, any output procedure referenced by that MERGE statement must appear:

      (1) Totally within nonindependent segments, or

      (2) Wholly contained in a single independent segment.

   b. If a MERGE statement appears in an independent segment, any output procedure referenced by that MERGE statement must be contained:

      (1) Totally within nonindependent segments, or

      (2) Wholly within the same independent segment.

7. If the GIVING phrase is specified, all of the merged records on file-name-1 are automatically written on file-name-5 as the implied output procedure for this MERGE statement.

8. In the case of an equal compare, according to the rules for comparison of operands in a relation condition, on the contents of the data items identified by all the KEY data-names between records from two or more input files (file-name-2, file-name-3, file-name-4, ...), the records are written on file-name-5 or returned to the output procedure, depending on the phrase specified, in the order that the associated input files are specified in the MERGE statement.

Special Considerations:

1. Key data items must not be described with USAGE COMPUTATIONAL-4.

2. For information concerning the merging of records described with an OCCURS...DEPENDING clause, refer to the Variable-Length Records paragraph in the MERGE portion of Section IX in the COBOL User's Guide.

The MOVE statement is used to transfer data to one or more data areas in accordance with the rules for editing.

Format 1:

```
MOVE { identifier-1 } TO identifier-2 [ , identifier-3 ] ...
     { literal      }
```

Format 2:

```
MOVE { CORR          } identifier-1 TO identifier-2 [, identifier-3] ...
     { CORRESPONDING }
```

Syntax Rules:

1. Identifier-1 and literal represent the sending area; identifier-2, identifier-3,..., represent the receiving area.
2. CORR is an abbreviation for CORRESPONDING.
3. When the CORRESPONDING phrase is used, all identifiers must be group items.
4. An index data item cannot appear as an operand of a MOVE statement.

General Rules:

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

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

      N - Numeric. This includes any item whose PICTURE consists solely of characters from the set 9, S, V, and P. It also includes the figurative constant ZERO and any numeric literal.

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

- An editing clause (e.g., BLANK WHEN ZERO).
- A PICTURE containing any of the numeric editing characters Z * $, . + - CR and DB.
- A PICTURE containing at least one of the insertion character B, and not containing any As or Xs.

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

AN - Alphanumeric. An item whose PICTURE contains only characters from the set A, X, 9 treated as if all were Xs. It also includes nonnumeric literals and the figurative constants except for ZERO and SPACES.

AB - Alphabetic. An item whose PICTURE consists entirely of As. It also includes the figurative constant SPACES.

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

- It is illegal to move an NE, AE, the figurative constant SPACE, or an AB item to an N or NE item.
- It is illegal to MOVE an N, the figurative constant ZERO, or an NE item to an AB item.
- It is illegal to MOVE an N item whose implicit decimal point is not immediately to the right of the least significant digit to an AN or AE item.
- A DISPLAY-2 item can be moved only to a DISPLAY or DISPLAY-2 item. When a DISPLAY-2 item is a receiving item, the sending item must be described with USAGE DISPLAY or DISPLAY-2 or be a literal or figurative constant.
- All other elementary moves are legal and are performed according to the procedures given in Rule 3 below.

3. Any necessary conversion of data from one form of internal representation to another takes place during legal elementary moves, along with any editing specified for the receiving data item:

a. When an alphanumeric edited or alphanumeric item is a receiving item, alignment and any necessary space-filling takes place as defined under the JUSTIFIED clause and the Standard Alignment Rules. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled. If the sending item is described as being signed numeric, the operational sign will not be moved.

b. When a numeric or numeric edited item is the receiving item, alignment by decimal point and any necessary zero-filling takes place as defined under the Standard Alignment Rules, except where zeros are replaced because of editing requirements.

- When a signed numeric USAGE DISPLAY or USAGE COMP-4 data item is the receiving item, the sign of the sending item is placed in the receiving item. Conversion of the representation of the sign takes place as necessary. If the sending item is unsigned, a positive sign is generated for the receiving item.

- When a USAGE COMPUTATIONAL or a USAGE COMP-1, COMP-2, or COMP-3 data item is the receiving item, the sign of the sending item is placed in the implicitly signed receiving item. If the sending item is unsigned, the receiving item will be positive.

- When an unsigned numeric item is the receiving item, the absolute value of the sending item is moved and no operational sign is generated for the receiving item.

- When a data item described as alphanumeric is the sending item, data is moved as if the sending item were described as an unsigned numeric integer.

c. When a receiving field is described as alphabetic, justification and any necessary space-filling takes place as defined under the JUSTIFIED clause and the Standard Alignment Rules. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled.

4. Any MOVE that is not an elementary move is treated exactly as if it were an alphanumeric-to-alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In such a move, the receiving area will be filled without consideration for the individual elementary or group items contained within either the sending or receiving area, except as noted in the rules for the OCCURS clause.

5. If the CORRESPONDING option is used, selected items within identifier-1 are moved, with any required editing, to selected items within identifier-2. Items are selected by matching the data-names of items defined within identifier-1 with like data-names of items defined within identifier-2 according to the following rules:

a. At least one of the items must be an elementary item.

b. The respective data-names are the same including all qualifications up to but not including identifier-1 and identifier-2.

c. A MOVE CORRESPONDING statement must not reference items having level-numbers 66, 77, or 88, or items described with the USAGE IS INDEX clause.

d. Any data-names which are subordinate to identifier-1 or identifier-2 and which have REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clauses are ignored, as well as any data-names which are subordinate to the data-names that contain REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clauses.

This restriction does not prevent identifier-1 or identifier-2 from having REDEFINES or OCCURS clauses or from being subordinate to data-names having REDEFINES or OCCURS clauses.

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

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

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

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

6. The following chart presents the relationship between the legality of a MOVE and the General Rules (above) which affect the given category.

Category of Sending Data Items		Category of Receiving Data Item		
		Alphabetic	Alphanumeric Edited Alphanumeric	Numeric Integer Num. Noninteger Numeric Edited
Alphabetic		Legal/3c	Legal/3a	Illegal/2a
Alphanumeric		Legal/3c	Legal/3a	Legal/3b
Alphanumeric Edited		Legal/3c	Legal/3a	Illegal/2a
Numeric	Integer	Illegal/2b	Legal/3a	Legal/3b
	Noninteger	Illegal/2b	Illegal/2b	Legal/3b
Numeric Edited		Illegal/2b	Legal/3a	Illegal/2a

7. When a sending and a receiving item in a MOVE statement share a part of their memory areas, the result of the execution of such a statement is undefined.

The MULTIPLY statement is used to multiply numeric data items and to set the values of data items equal to the results.

Format 1:

```
MULTIPLY { identifier-1 } BY identifier-2 [ROUNDED]
         { literal-1    }

         [, identifier-3 [ROUNDED] ] ...

         [ON SIZE ERROR imperative-statement]
```

Format 2:

```
MULTIPLY { identifier-1 } BY { identifier-2 }
         { literal-1    }   { literal-2    }

          GIVING identifier-3 [ROUNDED]

         [, identifier-4 [ROUNDED] ] ...

         [ON SIZE ERROR imperative-statement]
```

Syntax Rules:

1. Each identifier must refer to a numeric elementary item, except that those identifiers appearing to the right of the word GIVING (Format 2) can refer to data items containing editing symbols.
2. All literals used must be numeric literals.
3. The maximum size of each operand is 18 decimal digits. The composite of operands (the hypothetical data item resulting from the superimposition of all receiving data items of a given statement, aligned on their decimal points), must not contain more than 18 digits.

General Rules:

1. When Format 1 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The value of identifier-2 is then replaced by the product. Similar action occurs for identifier-1 or literal-1 and identifier-3, etc.

2. When Format 2 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2 or literal-2 and the product is stored in identifier-3, identifier-4, etc.

3. Refer to the Common Options in Statement Formats paragraph in this section for uses of the ROUNDED and SIZE ERROR options and multiple results.

The NOTE sentence is used in the Procedure Division to include explanatory information which is produced on the output listing but not compiled.

General Format:

NOTE character-string.

Rules:

1. Any combination of characters from the allowable character set may follow the word NOTE as long as the COBOL rules of punctuation and word and literal formation are observed.

2. If a NOTE sentence is the first sentence of a paragraph, the entire paragraph is considered to be a note. The format rules for paragraph structure must be observed.

3. If a NOTE sentence appears as other than the first sentence of a paragraph, the explanatory information of which the note is composed must end with a period followed by a space.

4. Notes are produced on the reference listing but have no effect on the object program.

5. The word NOTE can appear only as the first word of a COBOL sentence.

The OPEN statement is used to initiate the processing of files. An OPEN statement also causes label checking/writing and other input-output functions to be performed.

General Format:

```
                ┌ INPUT  file-name-1  [WITH NO REWIND]                  ┐
                │    [, file-name-2  [WITH NO REWIND]]   ...             │
OPEN            │ OUTPUT file-name-3  [WITH NO REWIND]                   │ ...
                │    [, file-name-4  [WITH NO REWIND]]   ...             │
                └ I-O    file-name-5  [, file-name-6]  ...               ┘
```

Syntax Rules:

1. Each of the format choices (INPUT, OUTPUT, I-O) may be specified only once in an OPEN statement.
2. The I-O phrase pertains to mass storage files only.
3. The WITH NO REWIND phrase does not apply to mass storage processing.

General Rules:

1. The OPEN statement must not be applied to sort files or merge files, but must be applied to all other files. The OPEN statement for a file must be executed prior to the first READ, WRITE, or SEEK statement for that file.
2. A second OPEN statement for a file cannot be executed prior to the execution of a CLOSE statement for that file.
3. The OPEN statement does not obtain or release the first data record. A READ or WRITE statement must be executed to obtain or release, respectively, the first data record. Data cannot be moved to an input record area, nor can the input record area be tested or referenced in any way until the first READ statement has been executed for the file, unless an APPLY PROCESS AREA phrase has been specified for that input file.

4. If a label record is specified for the file, the label is processed according to the standard beginning label convention. The behavior of the OPEN statement when a label record is specified but not present, or when a label record is not specified but is present, is undefined. If specified by the USE statement, a user's label procedure is executed. The order of execution of these two processes is specified by the USE statement. If label records are indicated as present by a LABEL RECORDS clause, the user's beginning label procedure (if specified by a USE statement) is executed before or after (as indicated) checking but subsequent to writing the first label.

5. The NO REWIND phrase can be used only with a sequential single reel/unit. (Refer to the CLOSE statement.)

6. If the external medium for the file permits rewinding, the following rules apply:

   a. When the NO REWIND phrase is not specified, execution of the OPEN statement causes the file to be positioned at its beginning.

   b. When the NO REWIND phrase is specified, execution of the OPEN statement does not cause the file to be repositioned. Therefore, when the NO REWIND phrase is specified, the file must have been positioned at its beginning.

7. If an input file is designated with the OPTIONAL phrase in the FILE-CONTROL paragraph of the Environment Division, the object program causes an interrogation for the presence or absence of this file. If the file is not present, the first READ statement for this file causes the imperative-statement in the AT END phrase to be executed.

8. The I-O option permits the opening of a mass storage file for both input and output operations. Since this option implies the existence of the file, it cannot be used if the mass storage file is being initially created.

9. When I-O is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

   a. The label (if it exists) is checked in accordance with the standard conventions for input-output label checking.

   b. The user's beginning label procedure, if one is specified by the USE statement, is executed.

   c. The new label is written in accordance with the standard conventions for input-output label writing.

10. When processing mass storage files for which the access mode is sequential, the OPEN statement supplies the initial address of the first record to be accessed.

11. * An OPEN statement may not be executed as a part of a label procedure appearing in the declarative portion of a program.

The PERFORM statement is used to depart from the normal sequence of procedures to execute one or more procedures a specified number of times, or until a condition is satisfied, and then return to the normal sequence.

Format 1:

```
PERFORM procedure-name-1 [ { THRU    } procedure-name-2 ]
                         { THROUGH }
```

Format 2:

```
PERFORM procedure-name-1 [ { THRU    } procedure-name-2 ]
                         { THROUGH }

        { identifier-1 }
        { integer-1    }  TIMES
```

Format 3:

```
PERFORM procedure-name-1 [ { THRU    } procedure-name-2 ]
                         { THROUGH }

        UNTIL condition-1
```

Format 4:

```
PERFORM procedure-name-1 [ { THRU    } procedure-name-2 ]
                         { THROUGH }

        VARYING { identifier-1 } FROM { identifier-2 }
                { index-name-1 }      { index-name-2 }
                                      { literal-1    }
```

```
BY    { identifier-3 }  UNTIL  condition-1
      { literal-2    }

[AFTER { identifier-4 } FROM { identifier-5 }
       { index-name-3 }      { index-name-4 }
                             { literal-3    }

 BY    { identifier-6 }  UNTIL  condition-2
       { literal-4    }

 [AFTER { identifier-7 }
        { index-name-5 }

  FROM { identifier-8 }  BY   { identifier-9 }  UNTIL condition-3]]
       { index-name-6 }      { literal-6    }
       { literal-5    }
```

Syntax Rules:

1. Each procedure-name is the name of a section or paragraph in the Procedure Division.
2. Each identifier represents a numeric elementary item described in the Data Division. In Format 2 and in Format 4 with the AFTER option, each identifier represents a numeric item with no positions to the right of the assumed decimal point.
3. Each literal represents a numeric literal.
4. The words THRU and THROUGH are equivalent.

General Rules:

1. The range of a PERFORM starts with the first executable statement in procedure-name-1 and continues in logical sequence through the last executable statement of:
   a. Procedure-name-2 if specified, or
   b. Procedure-name-1 if procedure-name-2 is not specified.

2. When the PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1. An automatic return to the statement following the PERFORM statement is established as follows:

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

   The 'last statement' performed in all of the above cases must allow control to pass to the return mechanism. There is no necessary relationship between procedure-name-1 and procedure-name-2 except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. In particular, GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2, provided control eventually passes to the return mechanism of procedure-name-2. If it is desired to have two or more logical paths to the return mechanism, then procedure-name-2 must be the name of a paragraph consisting of the EXIT statement, to which all of these paths must lead.

3. When control passes to these procedures by means other than an implicit or explicit PERFORM statement and no related PERFORM is in progress, sequence of control will pass through the 'last statement' to the following statement as if no PERFORM statement mentioning these procedures existed.

4. The PERFORM statement for Formats 1 through 4 operates as follows, with Rule 3 above applying to all formats:

   a. Format 1 is the basic PERFORM statement. A return to the statement following the PERFORM is inserted after the 'last statement' as defined in Rule 2, and sequence control is sent to procedure-name-1 for execution once.

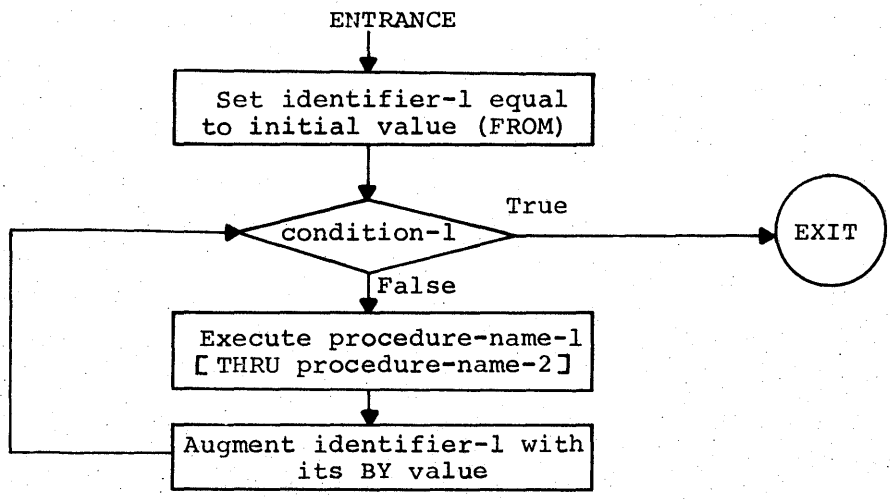
   b. Format 2 is the TIMES option. The specified number of times must be an integer. The integer may be negative or zero, in which case control passes to the next statement. The PERFORM mechanism sets up a counter and tests it against the specified value before each transfer to procedure-name-1. The return 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.

During execution of the PERFORM statement, reference to identifier-1 cannot alter the number of times the procedures are to be executed from that which was indicated by the initial value of identifier-1.

c. Format 3 is the UNTIL option. This option is similar to the TIMES option, except that an evaluation of a condition takes the place of counting and testing against a specified integer. The condition may be any simple or compound condition, as described under the Conditions paragraph in this section. When the condition is satisfied (true), control is transferred to the next statement after the PERFORM statement. If the condition is true when the PERFORM is entered, no transfer to procedure-name-1 takes place and control is passed to the next statement after the PERFORM statement.

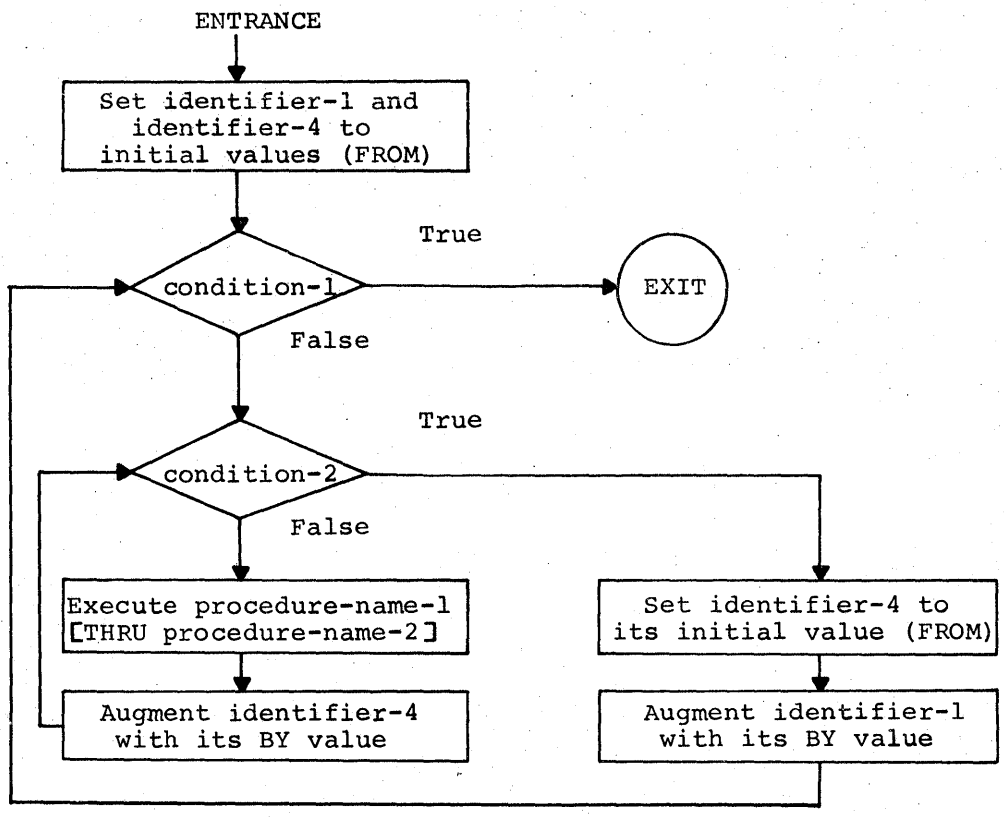
d. Format 4 is the VARYING option. This option is used to augment the value of one or more identifiers or index-names in an orderly fashion during the execution of a PERFORM statement. In the following discussion, every reference to identifier as the object of the VARYING and FROM (starting value) phrases also refers to index-names. When index-names are used, the FROM and BY phrases have the same effect as in a SET statement.

In Format 4, when one identifier is varied, identifier-1 is set to its starting value (the value of identifier-2 or literal-1) at the point of initial execution of the PERFORM statement; then, if the condition of the UNTIL phrase is false, the sequence of procedures, procedure-name-1 through procedure-name-2, is executed once. The value of identifier-1 is augmented by the specified increment or decrement value (the value of identifier-3 or literal-2) and condition-1 is evaluated again. The cycle continues until this condition is true, at which point control passes to the statement following the PERFORM statement. If condition-1 is true at the beginning of execution of the PERFORM statement, control passes directly to the statement following the PERFORM statement.



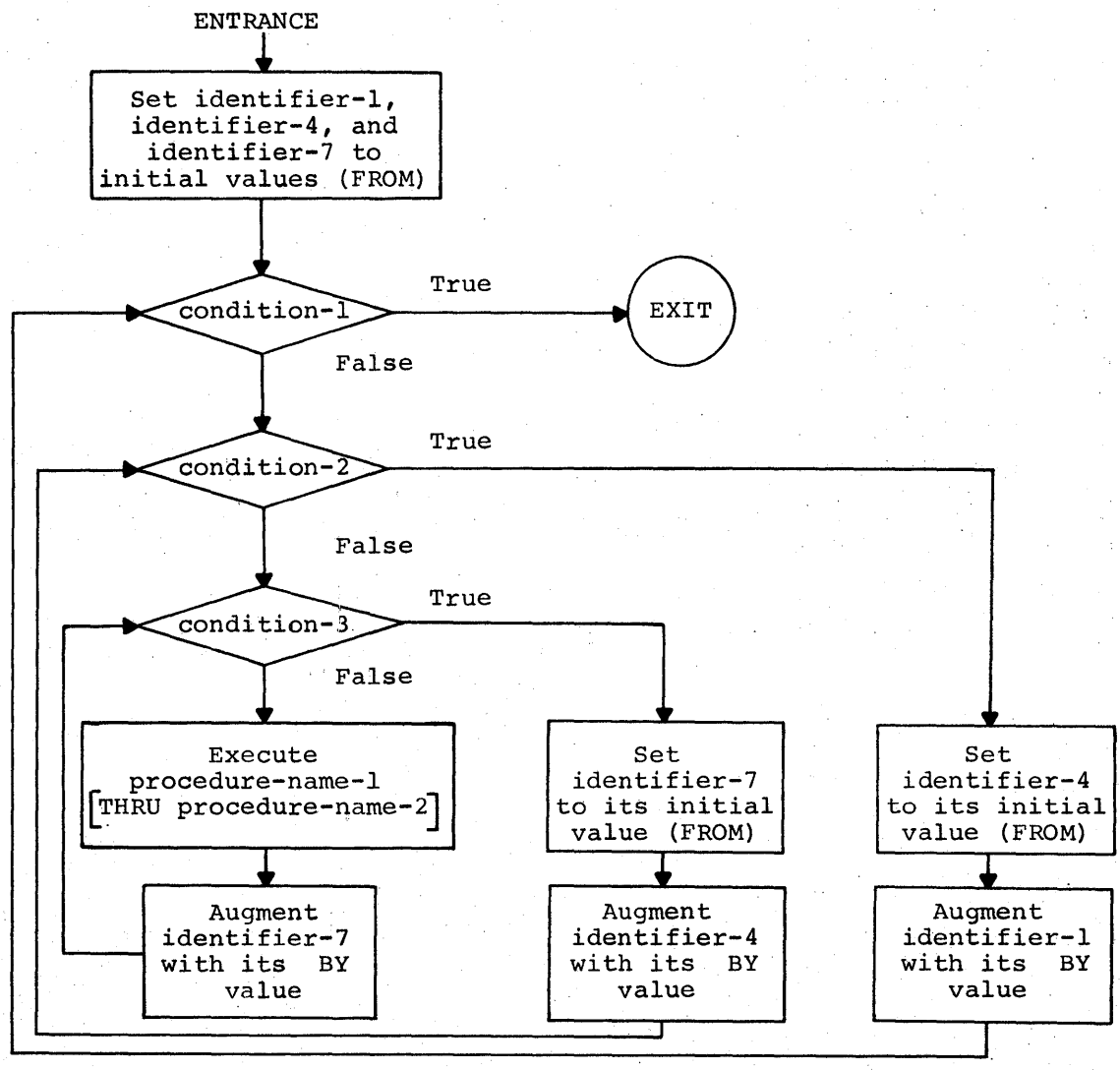
Flow Chart for the Varying Option of a PERFORM Statement Having One Condition

In Format 4, when two identifiers are varied, identifier-1 and identifier-4 are set to their initial values (the values of identifier-2 and identifier-5, respectively). During execution, these initial values must be positive. After initializing the identifiers, condition-1 is evaluated; if true, control is passed to the statement following the PERFORM statement; if false, condition-2 is evaluated. If condition-2 is false, procedure-name-1 through procedure-name-2 is executed once, then identifier-4 is augmented by identifier-6 or literal-4 and condition-2 is evaluated again. This cycle of evaluation and augmentation continues until condition-2 is true. When condition-2 is true, identifier-4 is set to its initial value (the value of identifier-5 or literal-3), identifier-1 is augmented by identifier-3 and condition-1 is re-evaluated. The PERFORM statement is completed if condition-1 is true; if not, the cycles continue until condition-1 is true. Identifier-3 and identifier-6 must not be zero. During execution of the PERFORM statement, reference to index-names or identifiers of the FROM phrase has no effect in altering the number of times the procedures are to be executed. Changing a value of index-names or identifiers of the VARYING phrase or identifiers of the BY phrase, however, will change the number of times procedures are executed.



Flow Chart for the Varying Option of a PERFORM Statement Having Two Conditions (Format 4)

At the termination of the PERFORM statement, identifier-4 contains its initial value, while identifier-1 has a value that exceeds the last used setting by an increment or decrement value, unless condition-1 was true when the PERFORM statement was entered, in which case identifier-1 and identifier-4 contain their initial values. When two identifiers are varied, identifier-4 goes through a complete cycle (FROM, BY, UNTIL) each time identifier-1 is varied. For three identifiers, the mechanism is the same as for two identifiers, except that identifier-7 goes through a complete cycle each time that identifier-4 is augmented by identifier-6 or literal-4, and identifier-4 in turn goes through a complete cycle each time identifier-1 is varied. The following flow chart illustrates the logic of the PERFORM statement when three identifiers are varied.



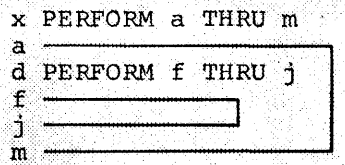
Flow Chart for the Varying Option of a PERFORM Statement Having Three Conditions

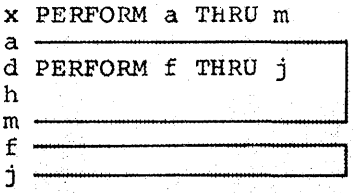
After the completion of the PERFORM statement, identifier-4 and identifier-7 contain their initial values, while identifier-1 has a value that exceeds its last used setting by one increment or decrement value, unless condition-1 is true when the PERFORM statement is entered, in which case identifier-1, identifier-4, and identifier-7 all contain their initial values.

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

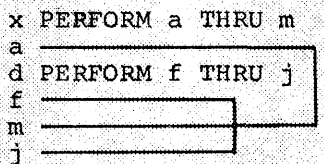
6. If a sequence of statements referenced by a PERFORM includes another PERFORM statement, the sequence associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referenced by the first PERFORM.

For example, the following usages are correct:





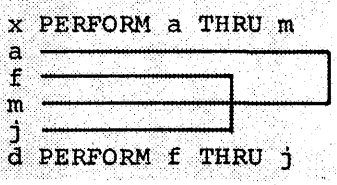
The following usage is incorrect:



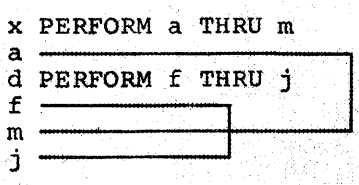
The sequence of procedures associated with a PERFORM statement may overlap or intersect the sequence associated with another PERFORM, provided that neither sequence includes the PERFORM statement associated with the other sequence.

For example:

Correct Usage



Incorrect Usage



7. A PERFORM statement that appears in a section whose priority-number is less than the SEGMENT-LIMIT, can have within its range only the following:

   a. Sections, each of which has a priority-number less than 50, or

   b. Sections wholly contained in a single segment whose priority-number is greater than 49.

8. A PERFORM statement that appears in a section whose priority-number is equal to or greater than the SEGMENT-LIMIT, can have within its range only the following:

   a. Sections, each of which has the same priority-number as that containing the PERFORM statement, or

   b. Sections having a priority-number less than the SEGMENT-LIMIT.

9. When a procedure-name in a segment with a priority-number greater than 49 is referred to by a PERFORM statement contained in a segment with a different priority-number, the segment referred to is made available in its initial state for each execution of the PERFORM statement.

For sequential-access file processing, the READ statement makes available the next logical record from an input or input-output file and allows performance of a specified imperative-statement when end-of-file is detected.

For random-access file processing, the READ statement makes available a specified record from a mass storage file and allows performance of a specified imperative-statement if the contents of the associated ACTUAL KEY data item are found to be invalid.

Format 1 (Sequential-Access Files):

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

    AT END imperative-statement
```

Format 2 (Random-Access Files):

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

    INVALID KEY imperative-statement
```

Syntax Rules:

1. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with 'identifier' and the storage area which is the record area associated with the file-name must not be the same storage area. File-name must not represent a sort file or a merge file.
2. Format 1 is used only for non-mass-storage files and for mass storage files in the sequential-access mode.
3. Format 2 is used for mass storage files in the random-access mode.

General Rules:

1. An OPEN statement must be executed for a file prior to the execution of the first READ statement for that file.

2. If after reading the last logical record of a file another READ statement is initiated for that file, that last logical record is no longer available in its record area and the READ statement is completed by the execution of the AT END phrase. Attempts to access the record area after the AT END phrase has been executed will cause unpredictable results. After the AT END condition has been recognized for a file, a READ statement for that file must not be given without prior execution of a CLOSE statement and an OPEN statement for that file. (Refer to Section V of the COBOL User's Guide.)

3. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. Only the information that is present in the current record is accessible.

4. If the INTO phrase is specified, the current record is moved from the input area to the area specified by identifier according to the rules specified for the MOVE statement without the CORRESPONDING phrase. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

5. When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.

6. If a file described with the OPTIONAL phrase is not present, the imperative-statement in the AT END phrase is executed on the first READ. The standard end-of-file procedures are not performed. (See the OPEN and USE statements, and the FILE-CONTROL paragraph in the Environment Division.)

7. If the end of a tape reel is recognized during execution of a READ statement, and the logical end of the file has not been reached, the following operations are accomplished:

   a. The standard ending reel label procedure and the user's ending reel label procedure, if specified by the USE statement. The order of execution of these two procedures is specified by the USE statement.

   b. A reel swap.

   c. The standard beginning reel label procedure and the user's beginning reel label procedure, if specified. The order of execution is again specified by the USE statement.

   d. The first data record of the new reel is made available.

8. Format 2 is used for mass storage files in the random-access mode. The READ statement implicitly performs the function of the SEEK statement for a specific mass storage file. If such files are * accessed for a specified mass storage record and the contents of the associated ACTUAL KEY data item are invalid, the INVALID KEY phrase is executed.

9. Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged in that a record is available prior to the execution of any statement following the READ statement.

10. A READ statement may not be executed as a part of a label procedure appearing in the declarative portion of a program.

The RELEASE statement is used to transfer records to the initial phase of a sort operation.

General Format:

RELEASE record-name [FROM identifier]

Syntax Rules:

1. The record-name must be the name of a logical record in an associated sort-merge file description entry and may be qualified.

2. The identifier and the record area associated with record-name must not be associated with the same storage area.

3. The RELEASE statement can be used only within the range of an input procedure associated with a SORT statement for a file whose sort-merge file description contains record-name. If the RELEASE statement is specified in any other application, the object program will terminate abnormally.

General Rules:

1. If the FROM phrase is specified, the contents of the identifier data area are moved to record-name and the contents of record-name are then transferred to the initial phase of the sort process. Moving takes place in accordance with the rules for the MOVE statement without the CORRESPONDING phrase. The information in the record area is no longer available but the information in the identifier area is available.

2. After the RELEASE statement is executed, the contents of record-name are no longer available.

3. When control passes from the input procedure, the sort file then consists of those records that were placed in it by the execution of RELEASE statements.

The RETURN statement is used to obtain ordered records from the final phase of a sort or merge operation.

General Format:

```
RETURN file-name RECORD [ INTO identifier ]

       AT END imperative-statement
```

Syntax Rules:

1. The file-name must be described in a sort-merge file description entry in the Data Division.
2. The identifier and the record area associated with file-name must not be associated with the same storage area.
3. The INTO phrase may be used only when the ordered file contains just one type of record.
4. The RETURN statement can be used only within the range of an output procedure associated with a SORT or MERGE statement for a file whose sort-merge file description contains file-name. If the RETURN statement is specified in any other application, the object program will terminate abnormally.

General Rules:

1. When successive RETURN statements are executed, the records are delivered in the order specified by the KEY data-names described in the SORT or MERGE statement. The next record becomes available for processing in the record area associated with file-name.
2. If the INTO phrase is specified, the current record is moved from the input area to the area specified by identifier in accordance with the rules for the MOVE statement without the CORRESPONDING phrase. When the INTO phrase is used, 'file-name RECORD' is still available in the input record area.
3. After the AT END imperative-statement is executed, a RETURN statement must not be executed within the current output procedure.

The SEARCH statement is used to search a table for a table element that satisfies the specified condition and to adjust the associated index-name to indicate that table element.

Format 1:

```
SEARCH identifier-1 [VARYING {index-name-1  }]
                             {identifier-2  }

    [AT END imperative-statement-1]

    WHEN condition-1 {imperative-statement-2}
                     {NEXT SENTENCE        }

    [WHEN condition-2 {imperative-statement-3}] ...
                      {NEXT SENTENCE        }
```

Format 2:

```
SEARCH ALL identifier-1 [AT END imperative-statement-1]

    WHEN condition-1 {imperative-statement-2}
                     {NEXT SENTENCE        }
```

Syntax Rules:

1. In both Formats 1 and 2, identifier-1 must not be subscripted or indexed, but its description must contain an OCCURS clause and an INDEXED BY phrase. The description of identifier-1 in Format 2 must also contain the KEY IS option in its OCCURS clause.

2. Identifier-2, when specified, must be described with USAGE INDEX or as a numeric elementary item with no positions to the right of the assumed decimal point. Identifier-2 is incremented by the same amount as, and at the same time as, the occurrence number represented by the index-name associated with identifier-1 is incremented.

3. In Format 1, condition-1, condition-2, etc., may be any condition as described previously in this section under Conditions.

4. In Format 2, condition-1 may consist of a relation condition incorporating the relation EQUALS or EQUAL TO or equal sign, or a condition-name condition, where the VALUE clause that describes the condition-name contains only a single literal. Alternatively, condition-1 may be a compound condition formed from simple conditions of the type just mentioned, with AND as the only connective. Any data-name that appears in the KEY phrase of identifier-1 may appear as the subject or object of a test or be the name of the conditional variable with which the tested condition-name is associated; however, all preceding data-names in the KEY phrase must also be included within condition-1. No other tests may appear within condition-1.

General Rules:

1. If Format 1 of SEARCH is used, a serial type of search operation takes place, starting with the current index setting. If the VARYING phrase specifying index-name-1 is used and index-name-1 occurs in the INDEXED BY phrase associated with identifier-1, index-name-1 specifies the index which controls the execution of the SEARCH statement. If the VARYING phrase is not used or does not specify an index-name-1 which occurs in the INDEXED BY phrase associated with identifier-1, the index which controls the execution of the SEARCH statement is specified by the first index-name that appears in the INDEXED BY phrase associated with identifier-1.

   a. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is greater than the highest permissible occurrence number for identifier-1, the SEARCH is terminated immediately. Then, if the AT END phrase is specified, imperative-statement-1 is executed; if the AT END phrase is not specified, control passes to the next sentence.

   b. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is not greater than the highest permissible occurrence number for identifier-1, the SEARCH statement operates by evaluating the conditions in the order that they are written, making use of the index settings, wherever specified, to determine the occurrence of those items to be tested. If none of the conditions are satisfied, the index-name for identifier-1 is incremented to obtain reference to the next occurrence. The process is then repeated using the new index-name settings unless the new value of the index-name settings for identifier-1 corresponds to a table element which exceeds the last element of the table by one or more occurrences, in which case the search terminates as indicated in a. above. If one of the conditions is satisfied upon its evaluation, the search terminates immediately and the imperative-statement associated with that condition is executed; the index-name remains set at the occurrence which caused the condition to be satisfied.

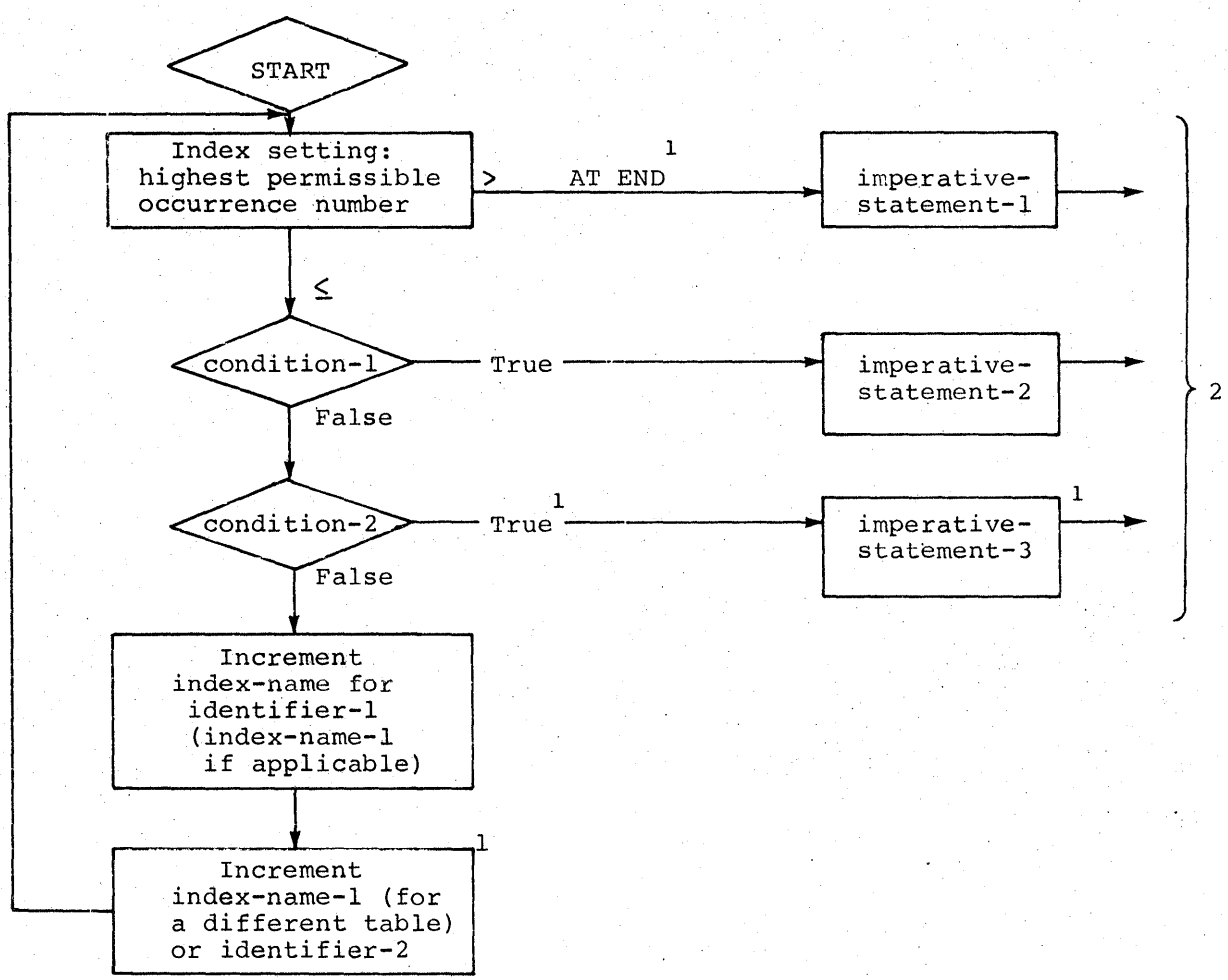
2. If Format 2 of SEARCH is used, a nonserial type of search operation takes place, in which case the initial setting of the index-name for identifier-1 is ignored and its setting is varied during the search operation in a manner which allows a 'binary' search operation to be executed, with the restriction that at no time is it set to a value that exceeds the value which corresponds to the last element of the table, or that is less than the value that corresponds to the first element of the table. The index that controls the execution of the SEARCH statement is specified by the first index-name that appears in the INDEXED BY phrase associated with identifier-1. If condition-1 cannot be satisfied for any setting of the index within this permitted range, control is passed to imperative-statement-1 when the AT END phrase appears, or to the next sentence when the AT END phrase does not appear; in either case the final setting of the index is not predictable. If condition-1 can be satisfied, the index indicates an occurrence that allows condition-1 to be satisfied, and control passes to imperative-statement-2.

3. After execution of imperative-statement-1, imperative-statement-2, or imperative-statement-3 that does not terminate with a GO TO statement, control passes to the next sentence.

4. In the VARYING index-name-1 phrase, if index-name-1 appears in the INDEXED BY phrase of another table entry, the occurrence number represented by index-name-1 is incremented by the same amount as, and at the same time as, the occurrence number represented by the index-name associated with identifier-1 is incremented.

5. If identifier-1 is a data item subordinate to a data item that contains an OCCURS clause (providing for a two- or three-dimensional table), an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Only the setting of the index-name associated with identifier-1 (and the data item identifier-2 or index-name-1, if present) is modified by the execution of the SEARCH statement. To search an entire two- or three-dimensional table, it is necessary to execute a SEARCH statement several times. Prior to each execution of a SEARCH statement, SET statements must be executed whenever index-names must be adjusted to appropriate settings.

A diagram of the Format 1 SEARCH operation containing two WHEN phrases follows:



[1] These operations are options included only when specified in the SEARCH statement.

[2] Each of these control transfers is to the next sentence unless the imperative-statement ends with a GO TO statement.

The SEEK statement is used to initiate the accessing of a mass storage data record for subsequent reading or writing.

General Format:

SEEK file-name RECORD

General Rules:

1. The SEEK statement pertains only to mass storage files in the random-access mode and may be present prior to the execution of each READ and WRITE statement. However, due to the requirements of the operating system, the SEEK statement has no effect on the access of a record.

2. Two SEEK statements for the same mass storage file may logically follow each other.

The SET statement is used to establish reference points for table-handling operations by setting index-names associated with table elements.

Format 1:

```
SET { index-name-1   [, index-name-2] ... } TO { index-name-3 }
    { identifier-1   [, identifier-2] ... }    { identifier-3 }
                                               { literal-1    }
```

Format 2:

```
SET index-name-4 [, index-name-5] ... { UP BY   } { identifier-4 }
                                      { DOWN BY } { literal-2    }
```

Syntax Rule:

1. All references to index-name-1, identifier-1, and index-name-4 apply equally to index-name-2, identifier-2, and index-name-5, respectively.

General Rules:

1. All identifiers must name either index data items or elementary items described as an integer, except that identifier-4 in Format 2 must not name an index data item. When a literal is used, it must be a positive integer. Index-names are considered related to a given table and are uniquely defined by being specified in the INDEXED BY phrase of the OCCURS clause.

2. In Format 1, the following action occurs:

   a. Index-name-1 is set to a value causing it to refer to the table element that corresponds in occurrence number to the table element referred to by index-name-3, identifier-3, or literal-1. If identifier-3 is an index data item, or if index-name-3 is related to the same table as index-name-1, no conversion takes place. If the value contained in an index data item does not correspond to an occurrence number of an element in the table indexed by index-name-1, the result is undefined.

   b. If identifier-1 is an index data item, it may be set equal to either the contents of index-name-3 or identifier-3, where identifier-3 is also an index data item. Literal-1 cannot be used in this case.

   c. If identifier-1 is not an index data item, it may be set only to an occurrence number that corresponds to the value of index-name-3. Neither identifier-3 nor literal-1 can be used in this case.

d. The process is repeated for index-name-2, identifier-2, etc., if specified. Each time, the value of index-name-3 or identifier-3 is used as it was at the beginning of the execution of the statement. Any subscripting or indexing associated with identifier-1, etc., is evaluated immediately before the value of the respective data item is changed.

4. In Format 2, the contents of index-name-4 are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of literal-2 or identifier-4; thereafter, the process is repeated for index-name-5, etc. Each time, the value of identifier-4 is used as it was at the beginning of the execution of the statement.

The SORT statement is used to create a sort file by executing input procedures or by transferring records from another file; to sort the records in the sort file on a set of specified keys; and, in the final phase of the sort operation, to make available each record from the sort file, in sorted order, to some output procedures or to an output file.

General Format:

```
SORT file-name-1 ON { DESCENDING } KEY data-name-1
                    { ASCENDING  }

    [, data-name-2 ] ...

[ ON { DESCENDING } KEY data-name-3 [, data-name-4 ] ... ] ...
     { ASCENDING  }

{ INPUT PROCEDURE IS section-name-1 [ { THRU    } section-name-2 ] }
{                                    { THROUGH }                 }
{       USING file-name-2                                        }

{ OUTPUT PROCEDURE IS section-name-3 [ { THRU    } section-name-4 ] }
{                                    { THROUGH }                 }
{       GIVING file-name-3                                       }
```

Syntax Rules:

1. File-name-1 must be described in a sort-merge file description entry in the Data Division. Each data-name must represent data items described in records associated with file-name-1.

2. Section-name-1 represents the name of an input procedure. Section-name-3 represents the name of an output procedure.

3. File-name-2 and file-name-3 must be described in a file description entry (not in a sort-merge file description entry) in the Data Division. The actual size of the logical record(s) described for file-name-2 and file-name-3 must be equal to the actual size of the logical record(s) described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, the corresponding records must be described in such a manner to cause equal amounts of memory to be allocated for the corresponding records.

4. The data-names may be qualified.

5. The words THRU and THROUGH are equivalent.

General Rules:

1. The KEY data-names are listed from left to right in the SORT statement in order of decreasing significance without regard to how they are divided into KEY phrases. In the format, data-name-1 is the major key and data-name-2 is the next most significant key.

   a. When an ASCENDING phrase is used, the sorted sequence will be from the lowest to the highest value of KEY according to the rules for comparison of operands in a Relation condition.

   b. When a DESCENDING phrase is used, the sorted sequence will be from the highest to the lowest value of KEY according to the rules for comparison of operands in a Relation condition.

2. When more than one record description entry appears in a sort file description, the key data items need be described only in one of the record description entries. Each key data item must occur in every data record of the sort file and must have the same relative position and actual format in all records. The PICTURE and USAGE of a given key data item must be the same for all records in the sort file. If a key data item is synchronized or justified, it must be identically synchronized or justified in all records in the sort file. The key data item descriptions must not contain an OCCURS clause or be subordinate to entries containing an OCCURS clause. Keys must be data items that do not require subscripting or indexing. The dominant record size is set equal to the size of the first record described in the sort file description entry (SD) in the Data Division. (See File Ordering - Sort and Merge in the COBOL User's Guide.)

3. If INPUT PROCEDURE is specified, control is passed to the input procedure before file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the input procedure; when control passes from the last statement in the input procedure, the records that have been released to file-name-1 will be sorted.

4. The input procedure, if present, must consist of one or more sections which must be written consecutively, and do not form a part of any output procedure. The input procedure must include at least one RELEASE statement in order to transfer records to the sort file. Control must not be passed to the input procedure except when a related SORT statement is being executed. The input procedure can include any procedures required to select, create, or modify records. Three restrictions apply to procedural statements within the input procedure:

   a. The input procedure must contain no SORT statements or MERGE statements.

b. The input procedure must contain no transfers of control to points outside the input procedure. That is, ALTER, GO TO, and PERFORM statements in the input procedure are not permitted to refer to procedure-names outside the input procedure. COBOL statements that cause an implied transfer of control to USE procedures are allowed.

c. The remainder of the Procedure Division must contain no transfers of control to points inside the input procedure. That is, ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division are not permitted to refer to procedure-names within the input procedure.

5. If the USING phrase is specified, all the records in file-name-2 are transferred automatically to file-name-1. When the SORT statement is executed, file-name-2 must not be open. The SORT statement will automatically perform the necessary OPEN, READ, and CLOSE functions for file-name-2. File-name-2 must have a file description entry (not a sort-merge file description entry) in the Data Division and must have the same file properties. The data records of file-name-2 and their descriptions must be identical to those of file-name-1.

6. If OUTPUT PROCEDURE is specified, control passes to the output procedure of the sort after file-name-1 has been sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the output procedure. When control passes from the last statement in the output procedure, the return mechanism provides for termination of the sort and then sends control to the next statement after the SORT statement. Before entering the output procedure, the sort can select the next record in sorted order when requested. The RETURN statements in the output procedure are the requests for the next record.

7. The output procedure, if present, must consist of one or more sections which must be written consecutively, and do not form a part of any input procedure. The output procedure must include at least one RETURN statement in order to make sorted records available for processing. Control must not be passed to the output procedure except when a related SORT statement is being executed. The output procedure can consist of any procedures required to select, modify, or copy the records which are being returned, one at a time in sorted order, from the sort file. Three restrictions apply to procedural statements within the output procedure:

a. The output procedure must contain no SORT statements or MERGE statements.

b. The output procedure must contain no transfers of control to points outside the output procedure. That is, ALTER, GO TO, and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedure. COBOL statements that cause an implied transfer of control to USE procedures are allowed.

c. The remainder of the Procedure Division must contain no transfers of control to points inside the output procedure. That is, ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division are not permitted to refer to procedure-names within the output procedure.

8. If the GIVING phrase is specified, all the sorted records in file-name-1 are automatically transferred to file-name-3 as the implied output procedure for this SORT statement. When the SORT statement is executed, file-name-3 must not be open. File-name-3 is automatically opened before transferring the records and a CLOSE file-name-3 is executed automatically after the last record in the sort file is returned. File-name-3 must have a file description entry (not a sort-merge file description entry) in the Data Division, with the same file properties. The data records of file-name-3 and their descriptions must be identical to those of file-name-1.

9. Segmentation, as defined in the COBOL User's Guide, can be applied to programs containing the SORT statement. However, the following restrictions apply:

   a. If a SORT statement appears in a section that is not in an independent segment, then any input procedures or output procedures referenced by that SORT statement must appear:

      1. Totally within nonindependent segments, or

      2. Wholly contained in a single independent segment.

   b. If a SORT statement appears in an independent segment, then any input procedures or output procedures referenced by that SORT statement must be contained:

      1. Totally within nonindependent segments, or

      2. Wholly within the same independent segment.

Special Considerations:

1. Key data items must not be described with USAGE COMPUTATIONAL-4.

2. For information concerning the sorting of records described with an OCCURS...DEPENDING clause, refer to the Variable-Length Records paragraph in the SORT portion of Section IX in the COBOL User's Guide.

The STOP statement is used to halt the execution of the run unit either temporarily or permanently.

General Format:

```
STOP { literal }
     { RUN     }
```

Syntax Rule:

1. The literal may be numeric, nonnumeric, or any figurative constant except ALL.

General Rules:

1. If the RUN option is used, the standard termination procedure is instituted for the current run unit.

2. If the STOP literal option is used, the literal is displayed upon the console display mechanism. When the operator acknowledges the message, continuation of the object program begins with the execution of the next statement in sequence. The literal used must conform to the rules for operands given for the DISPLAY statement. This option is not recommended except in unusual circumstances and should never be used to terminate the execution of the program.

3. If a STOP RUN statement appears in an imperative sentence, it must appear as the only or last statement in a sequence of imperative-statements.

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from one or more items, and set the values of one or more items equal to the results.

Format 1:

```
SUBTRACT  {literal-1   }  [, literal-2   ] ...
          {identifier-1}  [, identifier-2]

          FROM identifier-m [ROUNDED]

          [, identifier-n [ROUNDED]] ...

        [ ON SIZE ERROR imperative-statement]
```

Format 2:

```
SUBTRACT  {literal-1   }  [, literal-2   ] ...
          {identifier-1}  [, identifier-2]

          FROM {literal-m   }
               {identifier-m}

          GIVING identifier-n [ROUNDED]  [, identifier-o [ROUNDED]] ...

          [ON SIZE ERROR imperative-statement]
```

Format 3:

```
SUBTRACT  {CORR         }  identifier-1
          {CORRESPONDING}

          FROM identifier-2 [ROUNDED]

          [ON SIZE ERROR imperative-statement]
```

Syntax Rules:

1. Each identifier must refer to a numeric elementary item except:

   a. In Format 2, where the identifiers that appear only to the right of the word GIVING may refer to a data item that contains editing symbols.

   b. In Format 3, where each identifier must refer to a group item.

2. The maximum size of each operand is 18 decimal digits. The composite of operands (the hypothetical data item resulting from the superimposition of all operands of a given statement, excluding the data items that follow the word GIVING, aligned on their decimal points) must not contain more than eighteen digits.

3. CORR is an abbreviation for CORRESPONDING.

General Rules:

1. In Format 1, all literals or identifiers preceding the word FROM are added together and this total is subtracted from identifier-m, identifier-n, etc., and the differences are stored as the new values of identifier-m, identifier-n, etc.

2. In Format 2, all literals or identifiers preceding the word FROM are added together, the sum is subtracted from literal-m or identifier-m, and the result of the subtraction is stored as the new value of identifier-n, identifier-o, etc.

3. If Format 3 is used, data items in identifier-1 are subtracted from and stored into corresponding data items in identifier-2.

4. Refer to the Common Options in Statement Formats paragraph in this section for the uses of the ROUNDED, SIZE ERROR, and CORRESPONDING options, and multiple results.

The TERMINATE statement is used to terminate the processing of a report.

General Format:

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

Syntax Rule:

1. Each report-name specified in a TERMINATE statement must be defined by a report description entry in the Data Division.

General Rules:

1. TERMINATE produces all the control footings associated with this report as if a control break had just occurred at the highest level, i.e., FINAL control break, and completes the Report Writer functions for the named reports. The TERMINATE statement also produces the last PAGE FOOTING and/or REPORT FOOTING report groups for the named reports.

2. Appropriate PAGE and OVERFLOW HEADING and/or FOOTING report groups are prepared in their respective order for the report description entry.

3. A second TERMINATE for a particular report may not be executed unless a second INITIATE statement has been executed for the report-name. If a TERMINATE statement has been executed for a report, a GENERATE statement for that report must not be executed unless an intervening INITIATE statement for the report is executed.

4. If the ALL phrase is specified, all reports defined in the Report Section of the Data Division that were initiated are terminated.

5. TERMINATE does not close the file with which the report is associated. A CLOSE statement for the file must be executed after the TERMINATE statement has been executed. The TERMINATE statement performs Report Writer functions for individually described report programs analogous to the input-output functions performed by the CLOSE statement for individually described files.

6. SOURCE clauses in TYPE CONTROL FOOTING FINAL or TYPE REPORT FOOTING report groups refer to the values of the items during execution of the TERMINATE statement.

The USE statement specifies procedures for input-output label and error handling that are in addition to the standard procedures provided by the input-output system. It is also used to specify Procedure Division statements that are executed just before a report group named in the Report Section of the Data Division is produced.

The USE statement provides the mechanism for specifying out-of-line procedural statements for processing mass storage files.

Format 1:

```
USE AFTER STANDARD ERROR PROCEDURE ON { file-name-1 [, file-name-2] ... }
                                      { INPUT                        }
                                      { OUTPUT                       }
                                      { I-O                          }
```

Format 2:

```
USE { BEFORE } STANDARD [ BEGINNING ]  [ REEL ]
    { AFTER  }          [ ENDING    ]  [ FILE ]
                                       [ UNIT ]

    LABEL PROCEDURE ON { file-name-1 [, file-name-2] ... }
                       { INPUT                        }
                       { OUTPUT                       }
                       { I-O                          }
```

Format 3:

```
USE BEFORE REPORTING identifier-1
```

Syntax Rules:

1. A USE statement, when present, must immediately follow a section header in the declarative portion of the Procedure Division and must be followed by a period followed by a space. The remainder of the section must consist of one or more procedural paragraphs that define the procedures to be used.

2. If the file-name phrase is used as a part of Format 2, the file description entry for each file-name must not specify a LABEL RECORDS ARE OMITTED clause.

3. In Format 3, identifier-1 represents a nondetail report group named in the Report Section of the Data Division. An identifier must not appear in more than one USE statement.

   No Report Writer statement (GENERATE, INITIATE, or TERMINATE) may be written in a procedural paragraph or paragraphs following the USE sentence in the declarative portion.

4. The USE statement itself is never executed; rather, it defines the conditions calling for the execution of the USE procedures.

5. If the words BEGINNING or ENDING are not included in Format 2, the designated procedures are executed for both beginning and ending labels.

   If neither ▲ UNIT, REEL, nor FILE is included, the designated procedures are executed for both REEL and FILE labels. The REEL phrase is not applicable to mass storage files.

6. The same file-name can appear in a different specific arrangement of a format. However, the appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE declarative.

7. No file-name may represent a sort file or merge file.

General Rules:

1. The designated procedures are executed by the input-output system at the appropriate time as follows:

   a. In Format 1, after completing the standard input-output error routine.

   b. In Format 2, before or after a beginning or ending input label check procedure is executed.

      Before a beginning or ending output label is created.

      After a beginning or ending output label is created, but before it is written.

      Before or after a beginning or ending input-output label check procedure is executed.

      None of the Format 2 procedures will be performed for files that are being actively used as the USING or GIVING file in a SORT statement or a MERGE statement.

2. In Format 2, within the procedures of a USE declarative in which the USE statement specifies a phrase other than the file-name-1 phrase, references to common label items need not be qualified by a file-name. A common label item is an elementary data item that appears in every label record of the program, but at the same time does not appear in any data record of this program. Furthermore, a common label item must have the same name, description, and relative position in every label record.

USE

If the INPUT, OUTPUT, or I-O option is specified, the USE procedures do not apply, respectively, to input, output, or input-output files that are described with the LABEL RECORDS ARE OMITTED clause.

3. In Format 3, the designated procedures are executed by the Report Writer just before the named report group is produced, regardless of page, overflow, or control break associations with report groups. The report group may be any type except DETAIL.

4. Within a USE procedure, there must be no reference to nondeclarative procedures. Conversely, in the nondeclarative portion, there must be no reference to procedure-names that appear in the declarative portion, except that PERFORM statements may refer to a USE declarative having Formats 1, 2, or 3, or to the procedures associated with such a USE declarative.

5. In Format 2, label procedures must not execute any OPEN, CLOSE, SEEK, READ, WRITE, DISPLAY, DISPLAY UPON SYSOUT, or ACCEPT FROM GIN statements.

The WRITE statement is used to place a logical record on an output file. For mass storage files, the WRITE statement allows a specified imperative-statement to be executed if the contents of the associated actual key data item are found to be invalid.

Format 1:

```
WRITE record-name [FROM identifier-1]

[{BEFORE } ADVANCING {identifier-2 LINES          }]
 {AFTER  }           {integer LINES               }
                     {TO TOP OF PAGE              }
                     {mnemonic-name               }
```

Format 2:

```
WRITE record-name [FROM identifier-1]

      INVALID KEY imperative-statement
```

Syntax Rules:

1. The record-name must not represent a sort file or a merge file.
2. The record-name is the name of a logical record in the File Section of the Data Division and may be qualified.
3. Identifier-1 and the record area associated with record-name must not be associated with the same storage area.
4. When the ADVANCING phrase is specified, the file must be assigned with the FOR LISTING phrase in the SELECT sentence.
5. When identifier-2 is used in the ADVANCING phrase, it must be the name of a numeric elementary item described with no positions to the right of the assumed decimal point.
6. When integer is used in the ADVANCING phrase, it must have a positive or a zero value.
7. When mnemonic-name is used in the ADVANCING phrase, any mnemonic-name defined in the SPECIAL-NAMES paragraph is acceptable, although the mnemonic-name associated with the TOP IS phrase is preferred.

General Rules:

1. An OPEN statement must be executed for a file prior to the execution of the first WRITE statement for that file.

2. The logical record released by the execution of the WRITE statement is no longer available unless either a process area has been explicitly or implicitly applied on the file, or the associated file is named in a SAME RECORD AREA phrase. The logical record is also available to the program as a record of other files that are contained in the same SAME RECORD AREA phrase as the associated output file.

3. If the FROM phrase is specified, the value of identifier-1 is implicitly moved to record-name (the current output record area). Moving takes place in accordance with the rules for the MOVE statement without the CORRESPONDING phrase. After the WRITE statement is executed, the information in the identifier-1 area is available but the information in the record-name area is not available.

4. A WRITE statement may not be executed as a part of a label procedure appearing in the declarative portion of a program.

Format 1 (Serial Files):

1. For serial files assigned to magnetic tape, or for serial files that utilize the device independence feature of linked mass storage file space, the WRITE statement performs the following operations after recognizing an end-of-reel condition:

   a. The standard ending reel label procedures, and the user's ending reel label procedures, if specified by the USE statement. The order of execution of these procedures is determined by the USE statement.

   b. A reel swap. (Reel swap includes rewinding the completed reel and returning it to the standby condition.)

   c. The standard beginning reel label procedures, and the user's beginning reel label procedures, if specified by the USE statement. The order of execution of these procedures is determined by the USE statement.

2. The ADVANCING phrase provides control of the vertical positioning of each record (line) on a printed page of the listing. This phrase must not be used for a file described with an OCCURS...DEPENDING clause. If the ADVANCING phrase is not specified, automatic advancing will produce single spacing when the file is assigned with the FOR LISTING phrase in the SELECT sentence. If the ADVANCING phrase is specified, the automatic advancing is overridden as follows:

   a. If identifier-2 is used, the listing is advanced the number of lines equal to the current value associated with identifier-2.

   b. If integer is used, the value of integer determines the number of lines the listing will be advanced.

   c. If the TOP OF PAGE phrase is used, the listing is advanced to the top of the next page.

   d. If the mnemonic-name associated with the TOP IS phrase in the SPECIAL-NAMES paragraph is used, the listing is advanced to the top of the next page.

Format 2 (Mass Storage Files):

1. For files processed in the sequential-access mode, the imperative-statement in the INVALID KEY phrase is executed when the end of the file is reached and an attempt is made to execute a WRITE statement for that file. The end of the file is defined as being the physical end of the allocated mass storage unit.

2. For files processed in the random-access mode, the WRITE statement implicitly performs the function of the SEEK statement for a specific mass storage record. The imperative-statement in the INVALID KEY phrase is executed when the contents of the actual key being used to locate the mass storage record are found to be invalid. When an INVALID KEY condition exists, no writing takes place and the information in the record area is available for additional processing.

SECTION VIII

THE COBOL LIBRARY

DESCRIPTION OF THE COBOL LIBRARY

The library function provides the capability for specifying text that is to be copied from a library file. The COBOL library contains text that is available to a source program at compilation time. The effect of the compilation of library text is the same as if the text were actually written as part of the source program.

An entry in the COBOL library may contain source program text for the Environment Division, the Data Division, and the Procedure Division. The library is accessed using the COPY clause or the COPY statement.

COBOL library text is placed on the COBOL library as a function independent of the COBOL program.

COPY Clause

The COPY clause is used in association with a file-name or data-name in FD, SD, RD, record description, and report group description entries. The COPY clause is used to direct the compiler to duplicate text from a library into the source program or from an area defined elsewhere in the Data Division of a source program into another area of the Data Division.

Format 1:

```
COPY   library-name

[ REPLACING   word-1   BY   { word-2      }
                           { identifier-1 }
                           { literal-1    }

    [ , word-3   BY   { word-4      } ]   ... ].
                      { identifier-2 }
                      { literal-2    }
```

Format 2:

```
{level indicator}
{level-number   }   data-name-1  COPY  data-name-2  [FROM LIBRARY] .
```

Syntax Rules:

Format 1:

1. When the COPY clause is specified, the library-name is required. The library-name must be identical with the name associated with the desired text on the library.

2. In Format 1, a word is any COBOL word and may be one of the following:

   - Condition-name
   - Data-name
   - File-name
   - Mnemonic-name

3. In Format 1, the COPY clause may be specified only at the 01 level in data description entries and report group description entries.

Format 2:

1. When the information to be duplicated is on the library, the FROM LIBRARY phrase must be included.

2. When the FROM LIBRARY phrase is used, the name of the level 01 library entry must be included in the qualification of data-name-2 unless data-name-2 is itself a level 01 entry. This is required even if the level 01 library entry name is not necessary to make the reference unique.

General Rules:

Format 1:

1. Format 1 of the COPY clause represents the American National Standard COPY function and is engaged by including the LIBCPY option in the variable field of the $ COBOL card. The LIBCPY option may appear in any order in relation to other control card options.

2. The library text is copied from the library and the result of the compilation is the same as if the text were actually a part of the source program.

3. The COPY clause is printed on the annotated source program listing preceding the library text to which it refers.

4. The text contained on the library must not contain a COPY clause.

5. The COPY clause may appear:

   a. In any of the paragraphs of the Environment Division.

   b. In any level indicator entries or in an 01 level-number entry in the Data Division.

6. No other clause may appear in the same entry as the COPY clause with the exception of the Report Writer CODE clause which, if specified, must precede the COPY clause when a report description (RD) entry is to be copied.

7. The copying process is terminated by the end of the library text.

8. If format or syntax errors are encountered in source lines copied from a library, the compilation results are unpredictable. If the copied text contains source errors, the line number references given with the total errors message that follows the last source line in the compilation listing may not agree with the alter numbers in the compilation listing.

9. If the REPLACING phrase is used, each occurrence of word-1, word-3, etc., in the text being copied from the library is replaced by the word, identifier, or literal associated with it in the REPLACING phrase.

10. Use of the REPLACING phrase does not alter the text as it appears on the library.

11. The COPY clause is written in any of the following forms:

    a. Environment Division -

    ```
    SOURCE-COMPUTER.  copy-clause.
    OBJECT-COMPUTER.  copy-clause.
    SPECIAL-NAMES.    copy-clause.
    FILE-CONTROL.     copy-clause.
    I-O-CONTROL.      copy-clause.
    ```

    b. Data Division, File Section -

    ```
    FD  file-name       copy-clause.
    SD  sort-file-name  copy-clause.
    01  data-name       copy-clause.
    ```

    c. Data Division, Working-Storage Section -

    ```
    01  data-name  copy-clause.
    ```

    d. Data Division, Report Section -

    ```
    RD  file-name  [CODE mnemonic-name]  copy-clause.
    01  data-name  copy-clause.
    ```

Format 2:

1. Format 2 of the COPY clause represents the HIS COPY function and is engaged by including the COPY option in the variable field of the $ COBOL card. The COPY option may appear in any order in relation to other control card options.

2. The HIS COPY feature provides two options:

    a. Internal copy that permits duplication of text within the Data Division.

    b. Library copy, which is invoked by the FROM LIBRARY phrase, permits text to be copied from a library into the Data Division of a source program.

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

   If there are entries subordinate to data-name-1, it is the user's responsibility to ensure that the resulting hierarchical structure is correct. If the level-number of data-name-1 is 77, data-name-2 must be an elementary item.

4. If the level-numbers of data-name-1 and data-name-2 are both 01, any level 66 entries associated with the data-name-2 record description are inserted by the duplication process.

5. Data-name-2 may be qualified but not subscripted.

6. The HIS COPY clause can be written in any of the following forms:

   a. Data Division, File Section -

      FD file-name copy-clause.
      SD sort-file-name copy-clause.
      level-number data-name copy-clause.

   b. Data Division, Working-Storage Section -

      level-number data-name copy-clause.

   c. Data Division, Report Section -

      RD report-name [CODE mnemonic-name] copy-clause.
      level-number data-name copy-clause.

7. The copying process is terminated by the appearance of the Procedure Division header.

NOTE: Refer to the COBOL User's Guide for information on the use of the COPY clause with compressed source deck options.

COPY Statement

The COPY statement is used in the Procedure Division of a COBOL source program to incorporate paragraph procedures from the library into the source program with the capability for word substitution as text is copied.

Format 1:

```
{paragraph-name.                                    }
{section-name SECTION [priority-number] .           }

COPY library-name

[REPLACING word-1 BY {word-2      }
                     {literal-1   }
                     {identifier-1}

[, word-3 BY {word-4      } ] ... ].
             {literal-2   }
             {identifier-2}
```

Format 2:

```
paragraph-name. COPY library-name FROM LIBRARY.
```

NOTE: Paragraph-name and section-name are not part of the statement syntax; they are shown only for clarity.

Syntax Rules:

Format 1:

1. When the COPY statement is specified, the library-name is required. The library-name must be identical to the name associated with the desired text on the library.
2. In Format 1, a word is any COBOL word and may be one of the following:
   - Condition-name
   - Data-name
   - File-name
   - Mnemonic-name
   - Procedure-name

3. When the COPY statement is used, it must be the first statement in the paragraph or section.

Format 2:

1. When the information to be duplicated is on the library, the FROM LIBRARY phrase must be included.

2. Paragraph-names defined in the HIS COPY library must be unique.

General Rules:

Format 1:

1. Format 1 of the COPY statement represents the American National Standard COPY function and is engaged by including the LIBCPY option in the variable field of the $ COBOL card. The LIBCPY option may appear in any order in relation to other control card options.

2. The library text is copied from the library and the result of the compilation is the same as if the text were actually a part of the source program.

3. The COPY statement is printed on the annotated source program listing preceding the library text to which it refers.

4. The text contained on the library must not contain a COPY statement.

5. The COPY statement may appear only in the Procedure Division.

6. No other statement may appear in the same entry as the COPY statement.

7. The copying process is terminated by the end of the library text.

8. If format or syntax errors are encountered in source lines copied from a library, the compilation results are unpredictable. If the copied text contains source errors, the line number references given with the total errors message that follows the last source line in the compilation listing may not agree with the alter numbers in the compilation listing.

9. If the REPLACING phrase is used, each occurrence of word-1, word-3, etc., in the text being copied from the library is replaced by the word, identifier, or literal associated with it in the REPLACING phrase.

10. Use of the REPLACING phrase does not alter the text as it appears on the library.

11. The COPY statement cannot be used to duplicate section-names.

12. The COPY statement is written in the following form:

Procedure Division -

```
{paragraph-name.                                    }
{section-name  SECTION  [priority-number]  .        }  copy-statement.
```

Format 2:

1. Format 2 of the COPY statement represents the HIS COPY function and is engaged by including the COPY option in the variable field of the $ COBOL card. The COPY option may appear in any order in relation to other control card options.

2. In the HIS COPY function, library copy, which is invoked by the FROM LIBRARY phrase of the COPY statement, permits text to be copied from a library into the Procedure Division of a source program.

3. The HIS COPY statement is written in the following form:

Procedure Division -

```
    paragraph-name.  copy-statement.
```

NOTE: Refer to the COBOL User's Guide for information on the use of the COPY statement with compressed source deck options.

APPENDIX A

RESERVED WORDS

Reserved words are words that may be used in COBOL source programs but may not appear as user-defined words. The two types of reserved words are:

1. Keywords

   A keyword is one whose presence is required when the format in which the word appears is used in a source program. Within the formats shown in this manual, keywords are uppercase and underlined. The three types of keywords are:

   a. Verbs, such as ADD, READ, and ENTER.

   b. Required words, which appear in statement and entry formats.

   c. Words having a specific functional meaning, such as NEGATIVE, SECTION, TALLY, etc.

2. Optional Words

   Within each format, uppercase words that are not underlined are optional words. The presence or absence of optional words within a format does not alter the compiler's translation. Misspelling of an optional word, or using another word in its place, however, is not allowed.

The following list contains the COBOL reserved words; shading indicates nonstandard (American National Standard) reserved words.

600
6000
6000-EIS

ABOUT
ACCEPT
ACCEPT-TALLY
ACCESS
ACTUAL
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALPHANUMERIC
ALPHANUMERIC-EDITED
ALTER
ALTERNATE
AN
AND
APPLY
ARE
AREA(S)
ASCENDING
ASCII
ASSIGN
AT
AUTHOR

BCD
BEFORE
BEGINNING
BEGINNING-FILE-LABEL
BEGINNING-TAPE-LABEL
BINARY
BIT(S)
BLANK(S)
BLOCK
BLOCK-COUNT
BOTTOM
BY

CALL
CANCEL
CARD(S)
CD
CF
CH
CHARACTER(S)
CHECK
CLASS
CLOCK-UNITS
CLOSE
COBOL
CODE
COLLATE
COLUMN
COMMA
COMMERCIAL
COMMON
COMMUNICATION
COMMUNICATION-DEVICE
COMP
COMP-n
COMPILE
COMPUTATIONAL
COMPUTATIONAL-n
COMPUTE
CONFIGURATION
CONSOLE
CONSTANT
CONTAINS
CONTROL(S)
COPY
CORR
CORRESPONDING
COUNT
CURRENCY

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DAY-OF-WEEK
DE
DEBUG
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL
DECIMAL-POINT
DECLARATIVES
DEFINE
DEFINITIONS
DELETE
DELIMITED
DELIMITER
DENSITY
DEPENDING
DEPTH
DESCENDING
DESTINATION
DETAIL
DIGIT(S)
DISABLE
DISC
DISPLAY
DISPLAY-TALLY
DISPLAY-n
DIVIDE
DIVIDED
DIVISION
DOLLAR
DOWN
DRUM
DUPLICATES
DYNAMIC

EIS
ELECT
ELSE
EMI
ENABLE
END
END-OF-MESSAGE
END-OF-PAGE
END-OF-SEGMENT
END-OF-TRANSACTION
ENDING
ENDING-FILE-LABEL
ENDING-TAPE-LABEL
END-OF-FILE
END-OF-TAPE
ENTER
ENTRY
ENVIRONMENT
EOP
EQUAL(S)
ERROR(S)
ESI
ETI
EVERY
EXAMINE
EXCEEDS
EXCEPTION
EXIT
EXPONENTIATED
EXTEND

FD
FILE
FILE-CONTROL
FILE-LIMIT(S)
FILE-SERIAL-NUMBER
FILLER
FINAL
FIRST
FLOAT
FOOTING
FOR
FORMAT
FROM

GENERATE
GIN
GIVING
GLAPS
GMAP
GO
GREATER
GROUP
GTIME

HEADING
HIGH
HIGH-BOUNDS
HIGH-VALUE(S)
HMS
HOLD

I-O
I-O-CONTROL
ID
IDS
IDENTIFICATION
IF
IN
INDEX
INDEX-n
INDEXED
INDICATE
INITIAL
INITIALIZE
INITIATE
INPUT
INPUT-OUTPUT
INSPECT

INSTALLATION
INTO
INVALID
IS

JUST
JUSTIFIED

KEY(S)

LABEL
LABEL-DAY
LABEL-IDENTIFIER
LABEL-YEAR
LAST
LEADING
LEAVING
LEFT
LENGTH
LESS
LEVEL
LIBRARY
LIMIT(S)
LINAGE
LINAGE-COUNTER
LINE(S)
LINE-COUNTER
LINKAGE
LISTING
LOCATION
LOCK
LOW
LOW-BOUNDS
LOW-VALUE(S)
LOWER-BOUND(S)

MAGNETIC
MEMORY
MERGE
MESSAGE
MINUS
MODE
MODULES
MOVE
MULTIPLE
MULTIPLIED
MULTIPLY

NEGATIVE
NEXT
NO
NOT
NOTE
NUMBER
NUMERIC
NUMERIC-EDITED

OBJECT-COMPUTER
OBJECT-PROGRAM
OCCURS
OF
OFF
OH
OMITTED
ON
ONLY
OPEN
OPTIMIZE
OPTIONAL
OPTIONS
OR
ORGANIZATION
OTHERWISE
OUTPUT
OV
OVERFLOW
OVERLAY

PACKED
PAGE
PAGE-COUNTER
PERFORM
PF
PH
PHASE1
PIC
PICTURE
PK
PLACE(S)
PLUS
POINT
POINTER
POPUP
POSITION
POSITIVE
PREPARED
PRINTER
PRINTING
PRIORITY
PROCEDURE(S)
PROCEED
PROCESS
PROCESSING
PROGRAM
PROGRAM-ID
PROTECT
PROTECTION
PUNCH
PURGE-DATE

QUEUE
QUOTE(S)

RANDOM
RANGE
RD
READ
READER
RECEIVE
RECORD(S)
RECORD-COUNT
RECORDING
REDEFINES
REEL
REEL-NUMBER
REEL-SERIAL-NUMBER
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMARKS
REMOTE
REMOVAL
RENAMES
RENAMING
REPLACING
REPORT(S)
REPORTING
RERUN
RESERVE
RESET
RETENTION-PERIOD
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SA
SAME
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENT
SEGMENT-LIMIT
SELECT
SELECTED
SEND
SENTENCE
SENTINEL
SEPARATE
SEQUENCED
SEQUENTIAL
SERIAL
SET
SIGN
SIGNED
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SOURCE-ID
SPACE(S)
SPACE-SAVING
SPECIAL-NAMES
SPEED
STANDARD
START
STATEMENTS
STATUS
STORAGE
STOP
STRING
SUB-QUEUE-n
SUBTRACT
SUM
SUPERVISOR
SUPPRESS
SUSPEND
SWITCH
SYMBOL
SYMBOLIC
SYNC
SYNCHRONIZED
SYSOUT
SYSTEM

TABLE
TALLY
TALLYING
TAPE
TERMINAL
TERMINATE
TEXT
THAN
THEN
THROUGH
THRU
TIME(S)
TIME-SAVING
TO
TOP
TRAILING
TYPE
TYPEWRITER(S)

UNEQUAL
UNIT
UNITS
UNSTRING
UNTIL
UP
UPON
UPPER-BOUND(S)
USAGE
USE
USING

VLR
VALUE(S)
VARYING
WHEN
WITH
WORDS
WORKING-STORAGE
WRITE

ZERO(S)
ZEROES

+
-
*
/
**
>
<
=

INDEX

HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form

TITLE: SERIES 60(LEVEL 66)/6000 COBOL REFERENCE MANUAL ADDENDUM A

ORDER NO.: DD25A, REV. 0

DATED: FEBRUARY 1977

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

CUT ALONG LINE

PLEASE FOLD AND TAPE –
NOTE: U. S. Postal Service will not deliver stapled forms

FIRST CLASS
PERMIT NO. 39531
WALTHAM, MA
02154

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

Honeywell