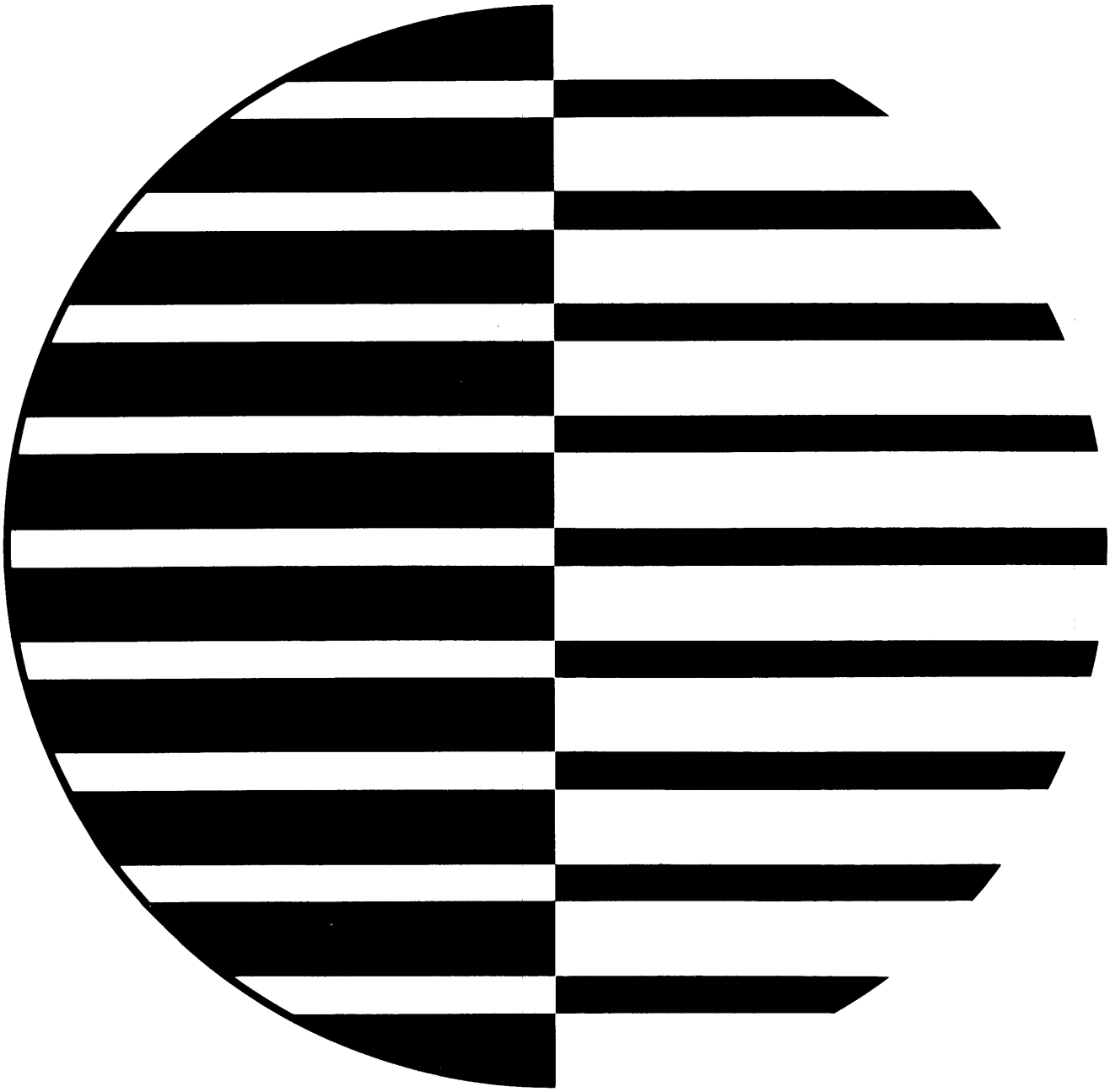
CONTROL DATA®
6400/6500/6600 COMPUTER SYSTEMS
COBOL Reference Manual



Additional copies of this manual may be obtained
from the nearest Control Data Corporation Sales office.

CONTROL DATA CORPORATION
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA

June, 1967
Pub. No. 60191200

ACKNOWLEDGEMENT

The following acknowledgment is in accordance with the requirements of the official government manual describing COBOL, Edition 1965.

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

Air Materiel Command, United States Air Force

Bureau of Standards, United States Department of Commerce

Burroughs Corporation

David Taylor Model Basin, Bureau of Ships, United States Navy

Electronic Data Processing Division, Minneapolis-Honeywell Regulator Co.

International Business Machines Corporation

Radio Corporation of America

Sylvania Electric Products, Inc.

UNIVAC Division of Sperry Rand Corporation

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

Allstate Insurance Company

The Bendix Corporation, Computer Division

Chase Manhattan Bank

Control Data Corporation

DuPont Corporation

General Electric Company

General Motors Corporation

Honeywell

Lockheed Aircraft Corporation

National Cash Register Company

Owens-Illinois Incorporated

Philco Corporation

Royal McBee Corporation

Space Technology Laboratories Incorporated

Southern Railway Company

Standard Oil Company (N.J.)

Sylvania Electric Products Incorporated

United States Steel Corporation

Westinghouse Electric Corporation

"Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken 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 this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention "COBOL" in acknowledgment of the source, but need not quote this entire section.

"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.

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

"The authors and copyright holders of 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."

CONTENTS

INTRODUCTION			vii
NOTATIONS USED IN MANUAL			ix
CHAPTER 1	IDENTIFICATION DIVISION		
	1.1	Specification of Identification Division	1-1
CHAPTER 2	ENVIRONMENT DIVISION		
	2.1	Specification of Environment Division	2-1
	2.2	Configuration Section	2-2
	2.3	Input-Output Section	2-4
CHAPTER 3	DATA DIVISION		
	3.1	Specification of Data Division	3-1
	3.2	Data Description	3-2
	3.3	Sections	3-5
	3.4	File Description Entry	3-8
	3.5	Record Description Entry	3-10
	3.6	Alphabetic List of Data Division Clauses	3-12
CHAPTER 4	PROCEDURE DIVISION		
	4.1	Specification of Procedure Division	4-1
	4.2	Declaratives	4-2
	4.3	Segmentation	4-2
	4.4	Statements and Sentences	4-4
	4.5	Conditions	4-5
	4.6	Arithmetic Expressions and Statements	4-14
	4.7	Options	4-16
	4.8	Alphabetic List of Procedure Division Statements	4-17
CHAPTER 5	THE REPORT WRITER		
	5.1	General Description	5-1
	5.2	Data Division Entry Formats	5-3
	5.3	Report Description Entry	5-4
	5.4	Report Group Description Entry	5-8
	5.5	Procedure Division Statements	5-21
	5.6	Sample Program	5-24

CHAPTER 6	SCOPE/COBOL INTERRELATION	
	6.1 Input-Output Control	6-1
	6.2 Compilation	6-10
	6.3 Execution	6-19
	6.4 COBOL Source Library	6-24
APPENDIX A	THE COBOL LANGUAGE	A-1
APPENDIX B	COBOL DISPLAY AND COLLATING SEQUENCE	B-1
APPENDIX C	COBOL RESERVED WORD LIST	C-1
APPENDIX D	INTERMEDIATE RESULTS IN ARITHMETIC EXPRESSIONS	D-1
APPENDIX E	CALLING SEQUENCE FOR ENTER	E-1
APPENDIX F	SAMPLE PROGRAMS	F-1
APPENDIX G	DIAGNOSTICS	G-1
APPENDIX H	CONVERSION HINTS	H-1
APPENDIX I	NUMBER REPRESENTATION	I-1
APPENDIX J	BINARY OUTPUT FROM COMPILER	J-1

INTRODUCTION

COBOL for the CONTROL DATA ® 6400, 6500, and 6600 computers is a language resembling English designed to simplify the programming of business data processing problems. Use of the COBOL language produces easily modifiable source programs resulting in shorter program development time and low program conversion costs.

This manual describes the 6400/6500/6600 COBOL source language, including a description of the Report Writer feature.

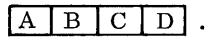
The 6400/6500/6600 COBOL source language is an upwards compatible subset of DOD COBOL, 1965, and is highly compatible with the CONTROL DATA 3000 series Compatible COBOL. In addition to all the features of the CONTROL DATA 3600 COBOL, the 6400/6500/6600 COBOL provides the following features:

- Mass Storage input and output
- SORT verb to sort files in conjunction with the 6400/6500/6600 Sort/Merge system
- RERUN option to allow restarting jobs at any specified point in the program
- RENAME option (level 66) to provide alternate naming of elementary items
- COMMON-STORAGE to permit data sharing by separately compiled programs
- COPY and INCLUDE to provide access to a source library
- REPORT WRITER to facilitate flexible formats for printed reports
- Segmentation and overlay of the object program
- 18-digit arithmetic operands
- Exponentiation available with COMPUTE
- Qualification allowed within the PROCEDURE division

The 6400/6500/6600 COBOL compiler is a two-pass system which resides on disk and operates under control of the SCOPE system. Programs generated by the compiler also operate under SCOPE control and use the input/output file manager, the sort, and the restart features provided by SCOPE.

NOTATIONS USED IN THIS MANUAL

Throughout this manual, basic formats are described for the essential elements of the COBOL language. These formats are intended to guide the programmer in writing statements according to the rules of the COBOL language. The following editorial conventions have been used.

- Material enclosed in square brackets [ ] may be included or omitted as required by the programmer.
- When material is enclosed in braces { }, one, and only one of the enclosed items must be chosen; the others are to be omitted.
- When a pair of braces or brackets is immediately followed by ... representing ellipses, the material within the braces or brackets may be repeated at the user's option.
- All words printed entirely in capital letters are COBOL words and have preassigned meaning to the COBOL processor.
- All underlined COBOL words are required unless the portion of the format containing them is itself optional. Such words are key words; if any is missing or misspelled, it is considered an error in the program. These words are not underlined by the programmer
- All COBOL words not underlined may be included or omitted at the option of the programmer. They are used only for readability. Misspelling such words when they are included in a format constitutes an error in the program.
- All words printed in small letters represent information which the programmer is to supply. These words generally indicate the nature of the information they represent (level-number, data-name).
- Special characters are essential where shown. For clarity, they are not underlined in the manual.
- Punctuation, where shown, is essential, and must be included by the programmer. Other punctuation marks may be included in accordance with the rules specified in this manual.
- The notation ↑ indicates the position of an assumed decimal point in an item.
- A numeric character with a plus or minus sign above it ($\overset{+}{n}$) indicates an operational sign is stored in combination with the numeric character.
- Character positions in storage are shown by boxes, 

. An empty box means an unpredictable result.
- Δ indicates a space (blank).

IDENTIFICATION DIVISION 1

The Identification Division specifies the information to identify the source program and the output from compilation. It must include the program name, and may also include the date the program was written, the date compiled, and so forth. Information specified in this division is included in the listing of the source program, but only the PROGRAM-ID clause affects the object program.

1.1 SPECIFICATION OF IDENTIFICATION DIVISION

```
IDENTIFICATION DIVISION .

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

The header IDENTIFICATION DIVISION begins in column 8 of the first line, and is followed by a period. The name of each succeeding paragraph is specified on a new line, each begins in column 8 and is followed by a period. Only the PROGRAM-ID paragraph is required.

PROGRAM-ID. program name.

Program name may be up to 30 alphanumeric characters; the first must be alphabetic. This name is used in referring to the source program, the object program, and all associated documentation. The first seven characters of the name are used by the SCOPE system to identify the program. When the subcompile capability is used, the first six characters in the program name must differ from the first six characters in the name of any other separately compiled subprogram that is part of the same COBOL program. The SCOPE system and subcompile capability are described in Chapter 6.

Succeeding paragraphs in this division supply documentary information. Any paragraph specified will appear in the source program listing.

ENVIRONMENT DIVISION 2

The Environment Division provides a method for describing any aspects of a COBOL program that depend on the physical characteristics of a specific computer. This division must be included in every COBOL source program.

The Environment Division must be rewritten and the entire source program recompiled when the object program is to be run on different computers. Rewriting and recompilation may also be necessary for different configurations of the same computer.

2.1 SPECIFICATION OF ENVIRONMENT DIVISION

The information in the Environment Division is specified in two sections:

The CONFIGURATION SECTION contains the computer specifications. The SOURCE-COMPUTER paragraph specifies the name of the compiling computer. The OBJECT-COMPUTER paragraph specifies the name of the executing computer. The SPECIAL-NAMES paragraph associates hardware devices, switch positions and character controls with mnemonic names in the source program. The INPUT-OUTPUT SECTION describes the external devices and the data storage techniques. The FILE-CONTROL paragraph names the files and their associated external devices. The I-O-CONTROL paragraph defines control techniques to be used in the object program.

```
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]
```

The header, ENVIRONMENT DIVISION begins in column 8 on the first line of the division. The section headers and paragraph names all must begin on new lines in column 8.

2.2 CONFIGURATION SECTION

The SOURCE-COMPUTER and OBJECT-COMPUTER paragraphs must be included; however, they are treated as notes and have no effect on the object program.

```
{ SOURCE-COMPUTER.   COPY library-name. }
{ SOURCE-COMPUTER.  {6400}              }
                    {6600}

{ OBJECT-COMPUTER.   COPY library-name. }
{ OBJECT-COMPUTER.  {6400}              }
                    {6600}
```

2.2.1 SPECIAL-NAMES

This paragraph equates mnemonic-names with special control characters and standard names of system files. It also equates switch-status-names with on or off status. Two clauses permit the specification of non-standard currency conventions. If none of these features is required, SPECIAL-NAMES may be omitted.

Format 1:

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

Format 1 is used only when the COBOL library contains the entire description of all SPECIAL-NAMES used in the program.

Format 2:

```
SPECIAL-NAMES.
[SWITCH integer-1
      { ON STATUS IS switch-status-name-1 [OFF STATUS IS switch-status-name-2] }]...
      { OFF STATUS IS switch-status-name-2 [ON STATUS IS switch-status-name-1] }

[implementor-name-1 IS mnemonic-name-1
      [implementor-name-2 IS mnemonic-name-2]...]

[CONSOLE IS mnemonic-name-3]

[CURRENCY SIGN IS literal]

[DECIMAL-POINT IS COMMA]

[non-numeric-literal-1 IS mnemonic-name-4
      [non-numeric-literal-2 IS mnemonic-name-5]...]
```

SWITCH

The SWITCH clause is used to test the ON/OFF status of a machine switch. SWITCH is followed by a space and an integer from 1 to 6 corresponding to switches 1 to 6. ON STATUS, OFF STATUS, or both may be specified for each switch depending on references in the Procedure Division. If a switch-status-name is not unique, it must be qualified by a switch name, SWITCH integer-1, whenever it is referenced. A reference to a switch-status-name in the Procedure Division is a reference to the corresponding position of the switch.

Switches are set by the operator or by control card as specified by SCOPE.

implementor-name IS mnemonic-name

This clause equates mnemonic names in the COBOL source program to valid SCOPE file names. The implementor name must conform to the naming conventions for SCOPE files (Chapter 6). All implementor names appearing in the Special Names paragraph and in the ASSIGN TO clauses of the File Control paragraph must be unique and may appear only once each. The special system files, such as INPUT and OUTPUT, are valid implementor name entries.

CONSOLE IS mnemonic-name

This clause equates a mnemonic name specified in the DISPLAY statement to the display console.

CURRENCY SIGN IS literal

The literal in this clause replaces the dollar sign wherever it appears in the PICTURE clause and the FLOAT DOLLAR SIGN wherever it appears in an editing clause. The literal is limited to a single character and it cannot be any of the following:

0 1 2 3 4 5 6 7 8 9

A B C D J K L P R S V X Z

* + - , ( ) .

DECIMAL-POINT IS COMMA

This clause exchanges the functions of the decimal point and the comma in the character string of PICTURE clauses and in numeric literals.

non-numeric-literal IS mnemonic-name

This clause is used to associated special control characters with mnemonic names referenced in the Procedure Division. It is required when a mnemonic name is specified in the WRITE BEFORE/AFTER option of the WRITE statement, and when the CODE option of the Report Writer is used. The non-numeric-literal is limited to one character, which must be enclosed in quotation marks. The character may be any one of the COBOL character set except the quotation mark.

2.3 INPUT-OUTPUT SECTION

This section consists of the header INPUT-OUTPUT SECTION and the two paragraphs FILE-CONTROL and I-O-CONTROL. If neither paragraph is needed, the entire section may be omitted.

2.3.1 FILE-CONTROL

This paragraph assigns implementor names to each file in the program. It may also be used to indicate that extra input-output areas are required for buffers.

Format 1:

FILE-CONTROL. COPY library-name.

Format 1 is used only when the COBOL library contains the entire description of the FILE-CONTROL paragraph.

Format 2:

FILE-CONTROL.

SELECT [OPTIONAL] file-name-1 [RENAMING file-name-2]
ASSIGN TO implementor-name
[FOR MULTIPLE { REEL / UNIT }]
[RESERVE { NO / integer-1 } ALTERNATE { AREA / AREAS }]
[FILE-LIMIT IS literal-1]
[ACCESS MODE IS { SEQUENTIAL / RANDOM }]
[PROCESSING MODE IS SEQUENTIAL]
[{ ACTUAL / SYMBOLIC } KEY IS data-name-5].
[SELECT...].

Each file described in the Data Division must be named and assigned to an implementor name in the FILE-CONTROL paragraph.

SELECT

The entry file begins with the word SELECT and terminates with a period. Each file named must be described in a DATA Division entry unless the RENAMING option is specified, and the file name must be the same as that used in the Data Division; this name must be unique within the source program. A single file name, however, may represent both an input and output file.

OPTIONAL

If an input file is named which will not necessarily be present when the object program is executed, the word OPTIONAL immediately precedes the file name following the word SELECT.

RENAMING

When file and record descriptions are identical for two files, they are written only once in the Data Division. (For example, a master file input to an updating run and the resulting new master file.) Then the RENAMING option is included in the FILE CONTROL paragraph. File-name-2 is the file described in the Data Division, file-name-1 is the file with the identical file and record descriptions. File-name-2 must be named elsewhere in the FILE CONTROL paragraph as the object of a SELECT entry; however, the SELECT clause must not contain a RENAMING option; and file-name-2 must not be a SORT file. This option does not permit interchangeable file names, each file name refers to a different file.

ASSIGN TO

A file named in a SELECT entry is assigned to an implementor name specified by this clause. Implementor-name must be a file name as specified by the SCOPE system (chapter 6). The special SCOPE system files, such as INPUT and OUTPUT, are valid entries for implementor-name. If such an association is specified, the system file name will become the external file name. The implementor-names appearing in the ASSIGN TO clause and in the Special-Names paragraph must be unique and each name may appear only once. The implementor name is formed like a data name except that it must be unique in the first six characters and may not contain a hyphen. A logical input-output device is assigned to the file named immediately following the word SELECT. If the RENAMING option is used, the file within that option (file-name-2) is assigned to a logical input-output device within its own SELECT entry.

A maximum of 53 files and reports can be specified in any one COBOL program. All files appearing in FD and SD entries must be named in this paragraph.

MULTIPLE REEL

May be specified for a file that uses more than one tape reel. This phrase is not necessary since multiple reels are handled automatically by the input-output control system which locates and assigns physical units as needed.

RESERVE ALTERNATE AREAS

Permits the user to specify additional buffer area. One buffer area is used per file for buffering data between the computer and the input-output devices. The one buffer area is circular; that is, the highest addressed location is immediately followed by the lowest. The size of this buffer may be increased by specifying additional alternate areas according to this formula:

S = (N*R) + (PRU-1)

where:

S = buffer size
N = number of alternate areas - 1
R = COBOL record size
PRU = physical record size, (see Chapter 6 for PRU)

FILE-LIMIT(S), ACCESS MODE, PROCESSING MODE, ACTUAL/SYMBOLIC KEY

These clauses may be specified for mass storage files only. ACCESS MODE is required for mass storage files. When the mode is SEQUENTIAL, mass storage records are obtained or placed sequentially. Neither the ACTUAL/SYMBOLIC KEY nor the FILE-LIMIT need be specified for a file with sequential access mode.

FILE-LIMIT must be used when the mode is RANDOM. It specifies the maximum number of records that can be written on or read from a random access file. Literal-1 becomes the size of the file index. The actual number of records written on a file determines when the INVALID KEY clause of a READ for that file is executed. The INVALID KEY clause for a WRITE is determined by the specified file limits.

When the mode is RANDOM, either ACTUAL KEY or SYMBOLIC KEY must be specified. The mass storage control system obtains or writes each record randomly. The specified logical record is located through data-name-5 of the ACTUAL/SYMBOLIC KEY clause and is made available on execution of a READ statement. When a WRITE statement is executed for a random access file, the record is effectively placed at the location in the file specified by data-name-5 of the ACTUAL/SYMBOLIC KEY clause. The user is responsible for setting the contents of data-name-5 prior to each READ, WRITE, or SEEK statement executed for a random access mass storage file. If no SEEK is issued prior to a READ or WRITE, the record is automatically located through the current value of data-name-5. If a SEEK is issued, the location of the record as specified by data-name-5 is made available to the next READ or WRITE executed. If data-name-5 names or points to the location of a record that is non-existent or outside the specified file limits, the INVALID KEY clause

of the READ or WRITE is executed. Data-name-5 of the ACTUAL/SYMBOLIC KEY clause must be unique; it may be qualified, but not subscripted. If ACTUAL KEY is specified, data-name-5 is a numeric item; at object time it contains a positive integer representing a position in the number index of a random access file. This index number is a digit from 1 through the maximum size for the file index specified by FILE-LIMITS. If SYMBOLIC KEY is specified, data-name-5 consists of one to seven display code characters representing the name in the name index associated with a record. The index tables resulting from this clause are described in chapter 6.

PROCESSING MODE IS SEQUENTIAL

This clause is for documentation purposes only. The sequential mode is used by the 6400/6500/6600 compiler to process all mass storage records. If both access and processing mode are sequential, records are requested and processed as they appear on the file. If access is random and processing is sequential, records may be requested in any order, and they are processed in the order requested.

2.3.2 I-O-CONTROL

This paragraph specifies the points at which rerun is to be made, the memory area to be shared by different files, and the location of files on a multiple file tape reel. If none of these techniques is needed, the I-O-CONTROL paragraph may be omitted.

Format 1:

I-O-CONTROL. COPY library-name.

Format 1 is used when the COBOL library contains the entire I-O-CONTROL paragraph. Otherwise format 2 must be used.

Format 2:

I-O-CONTROL.

[RERUN [ON file-name-1] EVERY { END OF REEL / integer-1 RECORDS } OF file-name-2]

[SAME [{ SORT / RECORD }] AREA FOR file-name-3, file-name-4... [file-name-5...]...]

[MULTIPLE FILE TAPE CONTAINS file-name-6 [POSITION integer-2] [file-name-7...
[POSITION integer-3...]]].

RERUN

This clause specifies complete checkpoint dumps at the end of each reel of file-name-2, or when the specified number of records is reached. Upon request for restart, all files are automatically repositioned and the program may be restarted at the most recent checkpoint. If file-name-2 is on a tape unit, either the REEL or the RECORDS option may be specified. If it is not on a tape unit, only the RECORDS option may be specified. If file-name-2 is a SORT file, the entire clause is ignored. File-name-1 is used for documentation only. The SCOPE system selects a disk file for the RERUN output.

SAME

This clause allows two or more files to share the same record area in memory, and also the same input-output areas for buffering. If the RECORD/SORT option is omitted, both the record and input-output areas will be shared by the files specified. If the RECORD option is specified, only the record area is shared; it is the area in storage affected by READ and WRITE statements.

SAME SORT AREA may be specified for documentation purposes only; it has no effect on the object program, since the sort/merge system uses the same area for all files sorted.

Since the opening of a file initializes the input-output and record areas, it is illegal for more than one of the files specified in a SAME clause to be open at the same time.

If two files specified in a RENAMING clause share the same record area, any reference to an item in the record area must be qualified by the name of the appropriate file.

The SAME clause may be repeated as necessary.

MULTIPLE FILE

This clause is required when two or more files share the same reel of tape. Regardless of the number of files on a single reel, only those files that are used in the object program need be specified. If all files on a reel are listed in the exact order that they will be read or written with no files omitted, the POSITION need not be given. Otherwise, POSITION is necessary to specify the position relative to the beginning of the tape for all the files to be processed. If POSITION is used for one file, it must be used for all other files on the reel.

DATA DIVISION 3

The Data Division, required in every COBOL program, contains a full description of data to be processed by the object program. Every item referenced by name in the Procedure Division, except the special registers and figurative constants (Appendix A), must be defined in the Data Division. An item is a specific area in memory which is named and defined in this division and which contains or will contain the data to be processed. Data may be divided into five types:

- Data stored or to be stored on an external device in the form of a file
- Data used to communicate between independently compiled programs
- Data developed internally during the operation of the program
- Data that has a fixed value
- Data to be output as reports

The Data Division, therefore, consists of File Section, Common-Storage Section, Working-Storage Section, Constant Section, and Report Section.

3.1 SPECIFICATION OF DATA DIVISION

```
DATA DIVISION.
FILE SECTION.
    .
    .
    .
COMMON-STORAGE SECTION.
    .
    .
    .
WORKING-STORAGE SECTION.
    .
    .
    .
CONSTANT SECTION.
    .
    .
    .
REPORT SECTION.
    .
    .
    .
```

The header DATA DIVISION is followed on a new line by a section header. Descriptions of elements involved in processing begin on the next line. These are followed by the next section header, and so forth. Each header is specified on a separate line, beginning in column 8, and is terminated by a period.

If data comprising a particular section is not required by the program, the section header must be omitted. When used, the sections must follow the order specified above.

3.2 DATA DESCRIPTION

3.2.1 ENTRY

The basic unit of description for the data in all sections is an entry. Each entry consists of a level indicator or level number, a name which can be referenced elsewhere in the program, and one or more clauses describing the data item.

Entries in the Data Division are of three types:

- File description entries describe physical characteristics of a file.
- Record description entries describe characteristics of items used in the program.
- Report, report group, and report element description entries describe items that are to appear on a report.

All sections except the Report Section contain record descriptions; the File Section contains file descriptions and record descriptions; the Report Section contains report descriptions only.

3.2.2 GROUP ITEM

A group item consists of two or more related items. A record description entry is written for each of the items in the group and the group itself must have at least a level number and a name. A group item may itself be part of a larger group. The most inclusive group item is called a record and must have the level number 01. (A record at the 01 level may be a single item.) When a record is in the Working-Storage, Common-Storage, or Constant Section, each name associated with the level number 01 must be unique. When a record is in the File Section the record name may be qualified by the file name. Data names of group items not at the 01 level need not be unique if they can be made unique by qualification.

3.2.3 ELEMENTARY ITEM

Elementary items, items that cannot be further subdivided, are preceded by a level number (2-49) which must be the largest number in the group containing the item.

3.2.4 INDEPENDENT ITEM

An independent item is a single item, which does not include any items that can be referenced independently, and is not contained in a larger item that can be referenced so as to include it. Each independent item requires a separate record description entry consisting of level number, data name, and descriptive clauses. Entries for independent items have the special level number 77. Level 77 items cannot occur in the File Section or in the Report Section.

3.2.5 LEVEL NUMBER

COBOL data description is based on the concept of levels of data. The more inclusive an item, the higher its level. Each entry in a COBOL program has a level indicator or level number. A file, which is the highest level of data, has a special level indicator, FD or SD, preceding the file name in a file description entry. The record is the next highest level of data, and a record description entry always has the level number 1 or 01 preceding the data name for the entry. A record may contain group items which may in turn contain group items or elementary items. Group items within a record and elementary items must have level numbers in the range 2-49. Level numbers need not be consecutive as long as the less inclusive item has the higher number. The specific level number is determined by the user.

Example: (This example gives only level number and data name; more clauses are required for a complete record description entry.)

```
01  PAYROLL-RECORD
    02  NAME
            04  FIRST-INITIAL
            04  SECOND-INITIAL
            04  LAST-NAME
    02  EMP-NUMBER
    02  DATE
            03  MONTH
            03  DAY
            03  YEAR
    02  HOURS-WORKED
```

3.2.6 SPECIAL LEVEL NUMBERS

Three special level numbers, 77, 88, and 66, specify particular types of record description entries rather than the hierarchy of the item. Level 77 identifies independent items in the Common-Storage, Working-Storage, and Constant Sections. Level 88 identifies record description entries for condition names, which may appear in the Data Division anywhere except the Constant Section. Level 66 is used to rename elementary items or group items. A level 66 item cannot be used to rename an item in another level 66 entry nor can it rename a 77, 88, or 01 level entry. A level 66 entry requires the RENAMES clause.

3.2.7 DATA-NAME

Every record description entry must have a subject; that is, a name assigned to the item. This name cannot be qualified or subscripted when used as the subject of a Data Division entry. The rules for forming data names, report names, file names and condition names are as follows:

The name may contain any of the characters: 0, 1, ...9,
A, B, ...Z,
- (hyphen)

Each name must contain at least one, but not more than 30 characters and at least one character in a data name must be alphabetic.

No name may contain a blank.

Names may neither begin nor end with a hyphen, although a hyphen may be used between other characters in the name for readability.

Two or more consecutive hyphens are not permitted.

3.2.8 FILLER

The word FILLER may be used instead of a data name as the subject of an entry that is never referenced. It may only name an unused elementary item. FILLER cannot be used to replace any of the following:

file name (FD)

report name (RD)

data name at the record level (01)

condition name or variable (88)

independent item name (77)

renamed item name (66)

An item assigned the name FILLER cannot be directly referenced from the Procedure Division, although a group containing a FILLER item can be referenced.

3.2.9 CONDITION NAME

A condition name is assigned to a value or set of values that a data item may assume. The record description entry for the data item is followed by one entry for each condition name associated with the item. A condition name must conform to the rules for a data name. A condition name entry is always preceded by the special level number 88 and must be followed by a VALUE clause specifying the particular value the data item may assume. Any reference in a conditional statement to the condition name becomes a reference to the value associated with it. Since each constant item has an initial unchanging value, condition names cannot be used in the Constant Section.

3.2.10 INITIAL VALUE

An initial value is the value of any item at the beginning of object program execution. This is specified by a VALUE clause in the record description entry in Working-Storage or Common Storage. If a value is not specified, the initial value of the item is unpredictable. Any value, numeric or non-numeric, may be assigned as the initial value of an item with the following restrictions:

The character used to specify the value must be in the same class as the item. Numeric items can have only numeric initial values; alphabetic items only alphabetic initial values.

If the number of character positions occupied by the initial value exceeds the number of character positions specified for the item in the SIZE clause, truncation will occur. If the size of the value is less, standard rules for justification apply.

Editing may not be performed on the initial value.

3.2.11 ASSIGNED VALUE

The value of a constant item is assigned by a VALUE clause in the entry. The value may be numeric or non-numeric, within the same restrictions indicated for the initial value of a working-storage or common-storage item. Every elementary and independent item in the Constant Section must have a value assigned unless it is within the scope of a REDEFINES.

3.2.12 LITERAL

Literals may be used wherever the format indicates. A literal is an explicit statement of the value to be used in an operation performed by the object program. Literals may be numeric or non-numeric. A complete definition of literals and the rules governing their formation is contained in Appendix A.

3.2.13 FIGURATIVE CONSTANT

Figurative constants are predefined as a part of the COBOL language. They may be used wherever a literal is allowed in the reference format. A complete list of figurative constants is contained in Appendix A.

3.3 SECTIONS

3.3.1 FILE SECTION

The File Section in the Data Division defines the contents of data files stored on an external device. One file description entry is specified for each file to be processed by the program. The section header is followed by the file description entry which consists of a level indicator (FD or SD), a file

name, and a series of independent clauses. Each file description entry is followed by record description entries for items associated with that file. These entries give the item patterns for logical records of different types and describe the characteristics of each item.

If two or more files are identical except for the file name, the file description and record description entries need not be repeated. These files are specified in the RENAMING clause of the FILE-CONTROL paragraph in the Environment Division. Sort files, however, may not appear in the RENAMING clause but must be specified individually in the File Section.

```
FILE SECTION.
FD file-name-1...
    01  data-name-1...
        02  data-name-2...
                  .
                  .
                  .
            03  data-name-3...
    01  data-name-4
        02  data-name-5
                  .
                  .
                  .
    01  data-name-6
SD file-name-2
    .
    .
    .
FD file-name-3
    .
    .
    .
```

3.3.2 COMMON-STORAGE SECTION

During execution of an object program, independently compiled subprograms may communicate through Common-Storage. Each item in a Common-Storage Section has a record description entry; all entries are preceded by the header COMMON-STORAGE SECTION and a period. The entries for independent items begin on new lines, and these are followed by entries for group items.

Each independently compiled subprogram which uses a Common-Storage Section for communication must define common-storage items in its Data Division. Data in the common-storage sections must be identical although the descriptions may differ. For instance, the same table fully described in one section might be alternately described with an OCCURS clause in the common-storage section of a separately compiled subprogram. The references from each subprogram differ, but the data referenced is the same.

The Common-Storage Section is allocated to a labeled common block named CCOMMON.

3.3.3 WORKING-STORAGE SECTION

During execution of an object program, intermediate results and other information need to be stored before being processed further or transferred out of memory. The storage areas, called Working-Storage items, are contained in the Working-Storage Section. Each item must have a record description entry. Entries are preceded by the header WORKING-STORAGE SECTION. Independent items are listed before grouped items.

```
{COMMON-STORAGE SECTION.}
{WORKING-STORAGE SECTION.}
    77  data-name-1
        88  condition-name-1
                :
                :
    77  data-name-2
    01  data-name-3
        02  data-name-4
                :
                :
        66  data-name-5 RENAMES data-name-4
    01  data-name-6
        02  data-name-7
            03  data-name-3
                88  condition-name-2
```

3.3.4 CONSTANT SECTION

The Constant Section of the Data Division contains all the record description entries specifying the named constants used in the program. A named constant is an item with a preassigned value that does not change during processing. The section header, CONSTANT SECTION and a period is followed on succeeding lines first by entries for independent constant items and then by entries for grouped constant items.

Every independent and elementary item in the Constant Section must have a value specified by a VALUE clause.

```
CONSTANT SECTION.
77  data-name-1...VALUE...
        :
77  data-name-2...VALUE...
01  data-name-3...VALUE...
    02  data-name-4...VALUE...
        :
01  data-name-5...VALUE...
    02  data-name-6...VALUE...
        03  data-name-7...VALUE...
```

3.3.5 REPORT SECTION

The Report Section is included when the Report Writer feature is used; it must be the last section in the Data Division. The header REPORT SECTION and a period is followed on succeeding lines by entries describing the physical format of each report to be produced. The level number RD and the report name are followed by clauses specifying the page structure. The report name must be named in the REPORT clause of the File Description Entry in the File Section. The Report Description Entry is followed by one or more report group entries, which, in turn, may include group and elementary entries. A Report Group Description entry consists of the level number 01, an optional data-name, and a series of independent clauses. Report Element Description entries, describing elementary or group items within a report group, consist of a level number (2-49), an optional data-name, and a series of independent clauses.

See Chapter 5 for a complete description of Report Writer entries.

```
REPORT SECTION.
RD report-name-1
    01  [data-name-1]
        02  [data-name-2]
            .
            .
            .
        02  [data-name-3]
    01  [data-name-4]
    01  [data-name-5]
            .
            .
            .
    01  [data-name-6]
RD report-name-2
            .
            .
            .
```

3.4 FILE DESCRIPTION ENTRY

The file description entries are at the highest level in the File Section. The section header is followed by a file description entry consisting of a level indicator (FD or SD), a file-name, and a series of independent clauses.

The file description entry for an external file (level FD) indicates recording mode, block size, labeling conventions, names of records or reports comprising the file, and so on. The object program requires this information to correctly interpret or create the file. This entry must be specified for each file.

The Sort File Description entry (level SD) is a special type of file description which gives information about the name, size and number of data records in the sort file. A sort file is defined as a set of records to be sorted by the SORT statement.

File Description Entry (FD):

FD file-name-1

Format 1:

COPY library-name.

Format 2:

[RECORDING MODE IS [{BINARY / DECIMAL}]] [{HIGH / LOW / HYPER} DENSITY]]

[FILE CONTAINS ABOUT integer-1 RECORDS]

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

[RECORD CONTAINS [integer-4 TO] integer-5 CHARACTERS [DEPENDING ON {RECORD-MARK / data-name-1}]]

LABEL {RECORDS ARE / RECORD IS} {STANDARD / OMITTED / data-name-2}

[VALUE OF {ID / IDENTIFICATION} IS {literal-1 / data-name-3}
[DATE-WRITTEN IS {literal-2 / data-name-4}]
[EDITION-NUMBER IS {literal-3 / data-name-5}]
[REEL-NUMBER IS {literal-4 / data-name-6}]
[RETENTION-CYCLE IS {literal-5 / data-name-7}]]

[VALUE OF ENDING-TAPE-LABEL-IDENTIFIER IS {literal-6 / data-name-8}]

{DATA {RECORDS ARE / RECORD IS} data-name-9 [data-name-10...]
{REPORTS ARE / REPORT IS} report-name-1 [report-name-2...].}

[SEQUENCED ON data-name-11 [data-name-12...]†

†This clause is used for documentation purposes only.

Sort File Description Entry:

SD file-name-2

Format 1:

COPY library-name.

Format 2:

[FILE CONTAINS ABOUT integer-1 RECORDS]

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

DATA $\left\{\begin{matrix}\text{RECORDS ARE}\\ \text{RECORD IS}\end{matrix}\right\}$ data-name-9 [data-name-10]... .

Specification of a File Description Entry

The level indicator (FD or SD) is written in columns 8 and 9; the file name, on the same line, starts in column 12. Clauses may follow on the same line separated from the file name by one or more spaces or on the next line starting in column 12. The order in which clauses are specified is unimportant; each clause is separated from the next by at least one space. No punctuation between clauses is necessary, but the entire entry must be terminated by a period. Each line after the first begins at or to the right of column 12.

The rules for continuing an entry on more than one line are given in Appendix A.

3.5 RECORD DESCRIPTION ENTRY

Every item referenced in a program must be described in a separate record description entry. Items are essentially specific areas in storage; the record description entries define the areas in terms of size, the manner in which they should be interpreted, and so on, based on the characteristics of the data to be stored in them.

Format 1:

level-number data-name-1 [REDEFINES data-name-2] COPY data-name-3 [FROM LIBRARY].

Format 2:

level-number { data-name-4 / FILLER } [REDEFINES data-name-5]

[SIZE IS integer-1 { CHARACTERS / DIGITS }]

[PICTURE IS character-string]

[USAGE IS { COMPUTATIONAL / COMPUTATIONAL-n / DISPLAY }]

[OCCURS [integer-3 TO] integer-4 TIMES [DEPENDING ON data-name-6]]

[SIGNED]

[SYNCHRONIZED { LEFT / RIGHT }]

[POINT LOCATION IS { LEFT / RIGHT } integer-5 [PLACES]]

[CLASS IS { ALPHABETIC / NUMERIC / ALPHANUMERIC / AN }]

[JUSTIFIED RIGHT]

[{ ZERO SUPPRESS / CHECK PROTECT / FLOAT DOLLAR SIGN / FLOAT CURRENCY SIGN } [LEAVING integer-6 PLACES]] [BLANK WHEN ZERO]

[VALUE IS literal-1].

Format 3:

66 data-name-7 RENAMES data-name-8 [THRU data-name-9].

Format 4:

88 condition-name { VALUE IS / VALUES ARE } literal-2 [THRU literal-3] [literal-4 [THRU literal-5]].

Specification of a Record Description Entry

Level numbers may be one or two digits. They begin in column 8 (01 or 1) or they begin to the right of column 11 (02 or 2 through 49).

With the exception of report groups and items, a data name must be specified on the same line as the level number and separated from it by at least one space, it should not start before column 12. The data name is optional in report groups and items.

The order of the clauses is unimportant, except where explicitly stated. The first clause is separated from the data name by at least one space, and each clause is separated from the next by at least one space. Punctuation between clauses is not necessary, but the entry is terminated by a period. The rules for continuing an entry on more than one line are given in Appendix A.

The specification of Report Description Entries is given in Chapter 5.

3.6 ALPHABETIC LIST OF DATA DIVISION CLAUSES

The following pages contain a complete list in alphabetic order of the clauses used in the File, Working-Storage, Common-Storage, and Constant Sections of the Data Division. Each clause is fully described.

$$\underline{\text{BLOCK}}\ \text{CONTAINS}\ [\text{integer-2 TO}]\ \text{integer-3}\left[\left\{\begin{matrix}\underline{\text{RECORDS}}\\ \text{CHARACTERS}\end{matrix}\right\}\right]$$

The basic input-output buffer size is specified in terms of logical records or characters. Logical records may be arranged one per block or more or less than one per block. If alternate areas are specified in the FILE-CONTROL paragraph of the Environment Division, this basic buffer size is increased by the number of alternate areas times the size specified in this clause.

Integer-2 and integer-3 are numeric literals with positive integral values. Integer-3 determines the maximum number of records or characters per block — the block size. Integer-2 may be used for documentation purposes but has no effect on block size. When record size is fixed, the word RECORDS is used; when record size is variable, the word CHARACTERS may follow integer-3 or may be omitted. If the clause is omitted, the block size is 512 words.

If a logical record is greater than 512 words, it is read or written in blocks of 512. It may be desired to specify a block size larger than record size so that an entire block may be read when only part of that block is needed to complete the record. The block size should not exceed the buffer size. In general, the buffer size should be:

$$S = (N*R) + (PRU-1)$$

where

S = buffer size
N = number alternate records - 1
R = COBOL record size
PRU = physical record size

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

Any data item can be classified as numeric, alphabetic, or alphanumeric. The CLASS clause indicates this classification to the processor.

NUMERIC	Items which consist only of digits 0-9; they may, however, also contain an assumed decimal point and an operational plus or minus sign, though none of these characters occupies a character position in the item in memory.
ALPHABETIC	Items which consist only of alphabetic characters and the blank (or space).
ALPHANUMERIC	Items which consist of any characters from the COBOL character set; numeric, alphabetic, and special characters (Appendix B).
AN	Acceptable abbreviation of ALPHANUMERIC.

The CLASS clause may be written for an item at any level. For a group item, it indicates the class for every item within that group and must not be contradicted by CLASS clauses within the group.

An ALPHANUMERIC group may contain both NUMERIC and ALPHABETIC items. A group referenced in a move operation should be described as ALPHANUMERIC. If class is omitted, or specified other than ALPHANUMERIC, group items are treated as ALPHANUMERIC when a move operation is performed.

A CLASS clause is not necessary when an entry contains a PICTURE clause, but if both are included, the PICTURE clause determines the class of the item; and the CLASS clause is used for documentation purposes only.

The class of all elementary items must be specified or implied; the class of 77 level items must be specified by either a PICTURE or CLASS clause.

The word CLASS is optional and may be omitted without altering the meaning of the clause. The word is omitted when SIZE, CLASS, and USAGE are combined.

Example:

```
01  INPUT-TABLE CLASS IS ALPHANUMERIC.
    03  PROG-NAME ALPHANUMERIC SIZE 10.
    03  LINES SIZE IS 5 NUMERIC COMPUTATIONAL-1 DIGITS.
    03  OCTALS.
        05  OCTAL-CHARS AN SIZE 1 OCCURS 6 TIMES.
```

Two editing clauses permit the programmer to specify editing without using the PICTURE clause. These clauses are used to describe report items, so that numeric data may be edited by moving it to such items; neither may be used with a PICTURE clause. All of the editing specified by such clauses can also be specified by the PICTURE clause.

Leading zeros may be suppressed and replaced in the edited data item by blanks or asterisks or blanks and one dollar sign. Zero suppression terminates with the first occurrence of a non-zero digit in the data or an assumed decimal point in the edited item whichever occurs first.

$$\left\{\begin{array}{l}\underline{\text{ZERO}}\ \underline{\text{SUPPRESS}}\\ \underline{\text{CHECK}}\ \text{PROTECT}\\ \underline{\text{FLOAT}}\ \text{DOLLAR}\ \underline{\text{SIGN}}\\ \underline{\text{FLOAT}}\ \text{CURRENCY}\ \underline{\text{SIGN}}\end{array}\right\}\ [\underline{\text{LEAVING}}\ \text{integer-6 PLACES}]$$

ZERO SUPPRESS

All leading zeros in data moved are suppressed and replaced by blanks in the edited data item.

CHECK PROTECT

All leading zeros are suppressed and replaced by asterisks in the edited data item.

FLOAT DOLLAR SIGN

All leading zeros are suppressed and replaced by blanks, except the rightmost which is replaced by a dollar sign in the edited data item.

FLOAT CURRENCY SIGN

Identical to FLOAT DOLLAR SIGN except that the replacement character is the currency sign specified in the Special Names paragraph.

LEAVING integer-6 PLACES

Terminates suppression before the assumed decimal point is encountered. Suppression of leading zeros terminates when integer-6 character positions remain to the left of the decimal point in the edited data item. Integer-6 must be a positive integer.

BLANK WHEN ZERO

This second editing clause is used for editing data items with a value of zero; it may be used in conjunction with the first clause. The edited data item is set to contain all blanks if the data value is zero. When the data is all zero and BLANK WHEN ZERO is specified, any editing, except check protect, specified in the first clause is overridden in favor of inserting blanks. If check protect is specified, the entire field is set to *; but if float dollar sign is specified with BLANK WHEN ZERO, the edited field contains all blanks with no dollar sign when the data value is zero.

Examples of Clause Editing:

Source Data	Editing Clause	Edited Item
0 0 3 4	ZERO SUPPRESS	Δ Δ 3 4
4 5 3	ZERO SUPPRESS	4 5 3
0 9 6	CHECK PROTECT	* 9 6
8 5 1	CHECK PROTECT	* 8 5 1
0 0 3 2	FLOAT DOLLAR SIGN	Δ $ 3 2
8 3 6 4	FLOAT DOLLAR SIGN	$ 8 3 6 4
0 0 0 0 9 5 8	ZERO SUPPRESS LEAVING 2 PLACES	Δ Δ Δ 0 9 5 8
0 0 0 0 0 2	ZERO SUPPRESS LEAVING 4 PLACES	Δ 0 0 0 0 2
0 0 0 0 3 8	CHECK PROTECT LEAVING 1 PLACE	* * * 0 3 8
0 0 0 1 4 7	FLOAT SIGN LEAVING 2 PLACES	Δ Δ $ 0 0 4 7
0 0 0 0 0 0	BLANK WHEN ZERO	Δ Δ Δ Δ Δ Δ

The receiving field must be large enough to contain the edited item.

This clause is used to copy data description entries from the COBOL source library, or from another part of the Data Division.

Format 1:

COPY library-name.

Format 1 is used to copy file description entries (FD and SD) when a complete description of the file exists in the COBOL library. It may also be used to copy the Special-Names, File-Control, or I-O-Control paragraphs in the Environment Division if a complete description of these files is in the COBOL library. Library-name is formed in the same manner as a data-name; it is up to 30 alphanumeric characters at least one of which must be alphabetic. When this format is used, no other description of a file is required.

Format 2:

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

Format 2 is used to copy record description entries from elsewhere in the Data Division or from a special library file. It eliminates the necessity of specifying such entries each time they occur.

Copying begins with the first clause in the data-name-2 entry; the clauses originally associated with data-name-2 are subsequently associated with data-name-1. All subsequent entries are copied in totality. Copying ends when an entry with a level number numerically equal to or less than the level number of data-name-2 is encountered, or when the end of the library entry is reached if the FROM LIBRARY option is used. Depending on the level structure of the information, one or more entries may be copied with a single COPY clause.

If there are items subordinate to data-name-1, the user must insure that the resulting hierarchical structure is correct. During the copying process, the level numbers of all inserted entries are adjusted by an amount equal to the difference between the level numbers of data-name-1 and data-name-2.

The copied entries may appear either before or after the entry referring to them. An entry being copied may not itself contain a COPY clause unless it is in the library. The copied entry may not contain a REDEFINES clause although the copying entry (data-name-1) may be redefined. Data-name-2 must not be the name of an item that requires subscripting. If the FROM LIBRARY option is specified, copying can be nested to a maximum of five levels.

Examples:

1. FD MASTER-FILE COPY FILEA.

   FILEA is the library-name of the COBOL source library deck containing a complete file description entry which will be copied into the source program as the description of the file named MASTER-FILE.

2. 01 SUM-DATA COPY SUMMARY-A.

If SUMMARY-A is a record description entry of the following form:

```
02 SUMMARY-A.
   03 COUNT PICTURE 9(3).
   03 G-TOTAL PICTURE 9(5)V99.
   03 O-TOTAL PICTURE 9(6)V99.
   03 G-DEVIATION PICTURE 9(4)V99.
   03 O-DEVIATION PICTURE 9(4)V99.
02 SUMMARY-B.
```

Then the following data description will be copied into the source program at the location of the COPY clause:

```
01 SUM-DATA.
   02 COUNT PICTURE 9(3).
   02 G-TOTAL PICTURE 9(5)V99.
   02 O-TOTAL PICTURE 9(6)V99.
   02 G-DEVIATION PICTURE 9(4)V99.
   02 O-DEVIATION PICTURE 9(4)V99.
```

DATA $\left\{\begin{matrix}\underline{\text{RECORDS}}\ \text{ARE} \\ \underline{\text{RECORD}}\ \text{IS}\end{matrix}\right\}$ data-name-3 [data-name-4...]

This clause must be specified for each file description entry. It permits the processor to correlate file and record description entries.

When more than one name is specified, the file contains a corresponding number of different types of logical records. The order in which the names are listed in the clause is not important.

The presence of more than one logical record type does not alter the basic concepts involved in handling individual logical records. Irrespective of type, all logical records from the same file are processed from a specific record area. The size of the record area is equivalent to the largest logical record in the file.

When the file description entry is for a sort file (SD), the data names identify records named in RELEASE statements.

This clause or the REPORT clause (Report Writer, Chapter 5), must be included in each File Description Entry. However, both clauses may not be specified in the same File Description Entry.

Examples:

```
FD CARD-FILE
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS CARD-INPUT.
FD MASTER-FILE
    LABEL RECORDS ARE OMITTED
    DATA RECORDS ARE DETAIL SUMMARY.
SD OUT-OF-SORT-FILE
    DATA RECORD IS INPUT-FILE.
```

FILE CONTAINS

FILE CONTAINS ABOUT integer-1 RECORDS

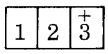
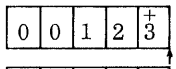
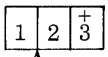
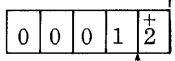
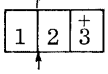
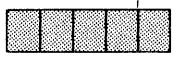
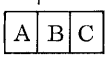
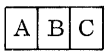
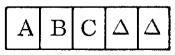
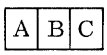
This clause indicates the approximate number of logical records in the file. It is optional and has no effect on the object program. Since it is printed as part of the source program listing, it provides documentation for this information.

JUSTIFIED RIGHT

This clause specifies positioning when a receiving item contains more or fewer character positions than there are in the data. Numeric data is aligned by decimal point. The justified clause, therefore, cannot be used on a numeric or edited item. Non-numeric data is left justified unless specified to the contrary by the JUSTIFIED RIGHT clause.

Justification occurs only when the data is transferred by any statement (except READ) that results in data movement.

Example:

Picture	Data	Item	Justified	
S9(5)			--	Right justified, zeros filled-in.
S9(4)V9			--	No justification, aligned by point.
S9(4)V9			Right	Illegal; item is numeric.
X(5)			Right	Right justified; blanks filled-in.
X(5)			--	Left justified normally.
X(2)			Right	Right justified, left character truncated.

Tape and disk files may contain label records which identify the file. A LABEL RECORD clause must be included in every file description entry regardless of the presence or absence of label records; it specifies whether labels are standard SCOPE labels, non-standard user defined labels, or omitted entirely. Only 1/2 inch coded magnetic tape files can have standard labels.

Format 1:

```
LABEL { RECORDS ARE } STANDARD VALUE OF { ID             } IS { literal-1 }
      { RECORD IS   }                   { IDENTIFICATION }    { data-name-2 }

                                        [DATE-WRITTEN      IS { literal-2 } ]
                                                              { data-name-3 }

                                        [REAL-NUMBER       IS { literal-3 } ]
                                                              { data-name-4 }

                                        [EDITION-NUMBER    IS { literal-4 } ]
                                                              { data-name-5 }

                                        [RETENTION-CYCLE   IS { literal-5 } ]
                                                              { data-name-6 }
```

Format 2:

```
LABEL { RECORDS ARE } data-name-7 [VALUE OF ENDING-TAPE-LABEL-IDENTIFIER IS { literal-6 } ]
      { RECORD IS   }                                                       { data-name-8 }
```

Format 3:

```
LABEL { RECORDS ARE } OMITTED
      { RECORD IS   }
```

Standard Label

If label records are standard, the first format is used. VALUE OF IDENTIFICATION must be included. For an input file, the operating system checks the label record for equality with the specified items. For an output file, it writes the specified items in the label record of the file. If DATE-WRITTEN is not included by the user for an output file, SCOPE provides the date. In either case, the date is converted and placed in the File Environment Table (Chapter 6) at the time an OPEN OUTPUT is executed for the file and prior to execution of any USE procedures for that file.

The data-names in the VALUE OF clause must be defined in the Common-Storage, Working-Storage, or Constant Sections. The items or literals specified in this clause must have the following characteristics:

[ID / IDENTIFICATION]	1-20 alphanumeric characters; the first must be alphabetic.
DATE-WRITTEN	6-digit integer, yymmdd = year, month, day
REEL-NUMBER	2-digit integer 01-99. If omitted, 01 is assumed. For subsequent reels of the file, this number is incremented by 1.
EDITION-NUMBER	2-digit integer, 01-99. If omitted, 00 is assumed.
RETENTION-CYCLE	3-digit integer, 000-999 specifying number of days from date-written that tape is to be saved; if omitted, 000 is assumed. 999 indicates indefinite retention.

Non-Standard Label

Format 2 is used when labels are non-standard. The data-name option indicates the first record on each reel is a separate physical record assumed to be a non-standard beginning label. Data-name-7 may not exceed 84 characters, but otherwise it is described like any other record in the file. For an input file, the first record is available to the user in the area specified by data-name-7 after an OPEN INPUT statement and before the first READ statement. The label record may be prepared before the OPEN OUTPUT statement for the file in the area defined by data-name-7. Non-standard label records may be processed and prepared with the USE statement.

VALUE OF ENDING-TAPE-LABEL-IDENTIFIER is used to distinguish between end-of-file and end-of-reel labels when a multiple-reel tape input file is being processed. This clause is used only for non-standard labels on multi-reel files; and it is effective only for input files. The literal or data name is 1 to 7 display characters. An end-of-reel label contains this value as the first characters of the ending label, but an end-of-file label does not. The value specified by literal-6 or data-name-8 is compared with the same number of characters at the beginning of the ending label to determine whether it is an end-of-file or an end-of-tape.

Memory contains a 40-character area called FILE-LABEL which may be referenced at any time from the Procedure Division; it is one area shared by all labels for all files. When labels are standard, this area corresponds to the standard label area of the File Environment Table (Chapter 6), and the format is identical to words 10-13 of the File Environment Table. When labels are non-standard, this area may be used to process or prepare file labels, and the user is responsible for the format of FILE-LABEL. Alternate names for FILE-LABEL are ENDING-FILE-LABEL, BEGINNING-FILE-LABEL, ENDING-TAPE-LABEL, and BEGINNING-TAPE-LABEL. These names are all interchangeable.

Omitted Label

Format 3 is used when a file has no labels or label processing is not required. LABEL RECORD IS OMITTED indicates no labels, and no information is transferred until the first READ statement for the file. Disk files must specify LABEL RECORD as data-name-7 if label processing or preparation is required, and LABEL RECORD IS OMITTED if not.

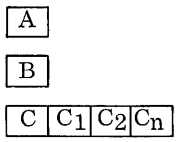
OCCURS [integer-3 TO] integer-4 TIMES [DEPENDING ON data-name-6]

When each item in a sequence is identical in every respect to the next one, except for value, the items need not be described in separate entries. The OCCURS clause indicates the number of times an item occurs and eliminates the necessity of repeating the description. Subscripting then permits an item in a sequence to be referenced by its position in the list. The manner in which an OCCURS clause is used is illustrated under the heading TABLES.

If the item occurs a fixed number of times, integer-4 represents the exact number of occurrences. If the exact number is not known, but it equals the value of a particular data item at object program execution, the integer-3 and the DEPENDING ON options are both included in the clause. The integer-3 and DEPENDING ON options do not indicate that an item is of variable length, only that the number of times it appears varies according to the value of another item.

The OCCURS...DEPENDING ON option may describe only one item in a record. No other items may follow this item in the record. The item described by OCCURS...DEPENDING ON may not be a conditional variable, nor may it be within the range of another OCCURS.

When a file contains a record with OCCURS...DEPENDING ON clause, the fixed portion of the record must be equal in length to all other records in the file, the key item (data-name-6) must be in the same relative position in the fixed portion of the record as any other key items in other records in the file, and the trailer items (occurring items) must all be equal in length



$A = B = C$

$C_1 = C_2 = C_n$

Integer-3 and integer-4 are numeric literals with positive integral values. When integer-3 is included it must be less than or equal to integer-4. Integer-3 may be zero, indicating that an item may not occur; if integer-3 is one, the item is present but may occur only once.

Data-name-6 is the name of an elementary item and it must be unique or made unique by qualification; but it may never be subscripted. This item can assume only positive integral values in the range defined by integer-3 and integer-4. The data-name-6 item must appear in Working-Storage, Common-Storage or in the same logical record as the entry containing the associated OCCURS clause and must precede the variable portion of the record. It must not appear in another file of the same program. An OCCURS clause may not appear in the same entry as a REDEFINES clause; but it may appear in an entry subordinate to an entry using REDEFINES.

Sequential items in an OCCURS clause can be referenced only by subscripting. If such items are groups, all references to items within the groups must be subscripted. Subscripting can be used only with items described by the OCCURS clause. All clauses in an entry that contains an OCCURS clause apply to each repetition of the item in the sequence. An item containing an OCCURS clause and SYNCHRONIZED clause will be synchronized at each occurrence. OCCURS may not be used with items at levels 01 and 77. RECORD CONTAINS...DEPENDING ON is used for records at 01 level in the File Section which contain a fixed portion and a variable number of trailer items. VALUE may not be specified in the OCCURS entry.

PICTURE IS any-allowable-combination-of-characters-and-symbols

Many of the clauses in a report description entry may be specified in an alternative, more compact manner with the PICTURE clause. The PICTURE clause specifies size and class, the presence or absence of an operational sign, assumed decimal point, and so on. It can specify editing of data (PICTURE EDITING). The PICTURE clause essentially specifies a picture of the data item, assembled from a set of code characters and symbols. Any item can be fully described using the PICTURE clause. Throughout this manual pictures are used to describe items which might in practice be described by other clauses.

Only elementary items can be described by the PICTURE clause. When PICTURE appears in an entry, it alone determines the description of the data item. If SIZE, CLASS, POINT LOCATION, Editing clauses (except BLANK WHEN ZERO) appear in the same entry, they are used for documentation only.

The number of occurrences of any of the characters indicates the size of an item described by the PICTURE clause. The size may be indicated either by repetition of the character, or in a shorthand way by writing the character once and putting the number of its occurrences in parentheses. Thus, P(10)9(2) is equivalent to PPPPPPPPPP99.

A maximum of 30 characters is allowed in a PICTURE clause. This limit does not refer to the number of characters in the item itself but only to the number of characters used in the picture specifying the item, including parentheses. For instance, the same item may be described by a picture containing 12 characters, PPPPPPPPPP99, or by a picture containing only 9 characters, P(10)9(2). In either case, the actual size of the item is only two characters. An item containing 75 alphabetic characters may be specified by the picture, A(75), which only uses 5 characters but the same item may not be specified by a picture in which A is repeated 75 times.

The size of an item described by the picture is limited to a maximum of 255 characters.

The characters and symbols in a PICTURE clause depend on the class of the data item: numeric, alphabetic, and alphanumeric.

Numeric Items

The PICTURE clause for a numeric item may contain only combinations of the following characters: 9 S V P.

9 This character indicates that the corresponding character position in the item will always contain a numeric character.

S This character indicates the presence of an operational sign, and is equivalent to the use of the SIGNED clause. When included, it is the leftmost character of the picture. An operational sign does not occupy a character position in the item, so it is not counted in the size of an item.

V This character indicates the position of an assumed decimal point for aligning items during computation. An actual decimal point can never appear in a numeric item. An assumed decimal point does not occupy a character position in the item and is not counted in the size of an item. The V is specified in the picture between the code characters which represent the characters on either side of the assumed decimal point in the item.

For example, if a data item is described as having a picture of 9V999 and it contains the digits 2567 at reference time, then the size of the item is considered to be four characters, and its value would be 2.567 for calculation, although it would be displayed as 2567 because the decimal point character is not actually present. A V directly to the right of all 9's is redundant.

P This character specifies the position of an assumed decimal point when this position does not occur within the characters that actually comprise the data item. If the assumed decimal point is to the right of the rightmost character in the item, one P is specified in the picture for each implied character position between the assumed decimal point and the rightmost character. Similarly, if the assumed decimal point is to the left of the leftmost character in the item, one P is specified in the picture for each implied character position between the assumed decimal point and the leftmost character. The item is treated as if a zero were substituted for each P and a decimal point placed to the right or to the left of the last P. The character P is never considered in determining the size of the item.

For example, an item composed of the digits 2567 is treated as 256700. in computations if the picture is 9999PP or as .002567 if the picture is PP9999.

If an entry contains a VALUE clause and a PICTURE clause, the literal in the VALUE clause must exactly reflect the picture even though a sign or decimal point will not actually be placed in the associated item.

For example, if an item is described by the picture SPP9999, and is initially to contain the characters 4513, the VALUE clause in the entry must be specified as VALUE IS .004513, or VALUE IS +.004513.

Alphabetic Items

The PICTURE clause for an alphabetic item may contain only the code character A, although this may be specified as many times as required.

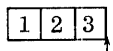
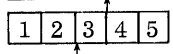
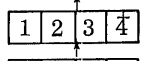
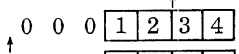
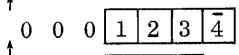
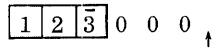
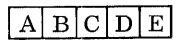
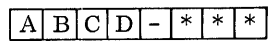
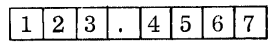
A This character indicates that the corresponding character position in the item will always contain either a letter of the alphabet or a blank (space).

Alphanumeric Items

Alphanumeric items which do not specify editing may contain only the characters X, 9, and A. The use of the characters 9 and A in alphanumeric items is the same as in numeric and alphabetic items.

X This character indicates that the corresponding character position in the item may contain any character in the COBOL character set. An A or 9 in an alphanumeric item, although not illegal, is treated exactly as an X; neither character need be used in the picture of such an item.

Examples:

Picture of Item	Characters in Item	Item
999	123	
99V999	12345	
S99V99	123^{+}_{4}	
PPP9999	1234	
SPPP9999	123^{-}_{4}	
S999PPP	$12\bar{3}$	
AAAAA or A(5)	ABCDE	
XXXXXXXX or X(8)	ABCD-***	
XXXXXXXX or X(8)	123.4567	

The table below shows the possible legal combinations of characters in a PICTURE clause. Diagnostic tests will be made for most of the conditions represented in the table. The characters A, X, 9 are described above; the remaining characters are described under PICTURE EDITING.

Given character in picture \ Is this character legal anywhere to right	A	X	9	S	V	P	Z	*	$	,	.	B	0	-	+	C	D	R	Repeated $	Repeated -	Repeated +
A	x	x	x	No	No	No	No	No	No	No	No	x	x	No	No	No	No	No	No	No	No
X	x	x	x	No	No	No	No	No	No	No	No	x	x	No	No	No	No	No	No	No	No
9	x	x	x	No	x	x	No	No	No	x	x	x	x	x	x	x	x	x	No	No	No
S	No	No	x	No	x	x	No	No	No	No	No	No	x	No	No	No	No	No	No	No	No
V	No	No	x	No	No	x	2	2	No	x	No	x	x	x	x	x	x	x	2	2	2
P	No	No	x	No	x	x	x	x	No	x	No	x	x	x	x	x	x	x	x	x	x
Z	No	No	x	No	x	x	x	No	No	x	x	x	x	x	x	x	x	x	No	No	No
*	No	No	x	No	x	x	No	x	No	x	x	x	x	x	x	x	x	x	No	No	No
$	No	No	x	No	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Yes	Yes
,	No	No	x	No	x	x	x	x	No	x	x	x	x	x	x	x	x	x	x	x	x
.	No	No	x	No	No	No	2	2	No	x	No	x	x	x	x	x	x	x	2	2	2
B	x	x	x	No	x	x	x	x	x	x	x	x	x	x	x	x	x	x	No	No	No
0	x	x	x	No	x	x	x	x	x	x	x	x	x	x	x	x	x	x	No	No	No
Leading -	No	No	x	No	x	x	x	x	x	x	x	x	x	No	No	No	No	No	x	x	No
Leading +	No	No	x	No	x	x	x	x	x	x	x	x	x	No	No	No	No	No	x	No	x
C	No	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No	No	No	x	No	No	No
D	No	No	No	No	No	Yes	No	No	No	No	No	x	No	No	No	No	No	No	No	No	No
R	No	No	No	No	No	1	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
Trailing -	No	No	No	No	No	1	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
Trailing +	No	No	No	No	No	1	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No

x Yes
1 Yes if all digits to the right are the same
2 Yes if all digits both to the right and left are the same

Editing alters the format and the punctuation of data in an item; characters can be suppressed or added.

Editing is accomplished by moving a data item to an item described as containing editing symbols. Movement may be direct or indirect. The programmer can specify a MOVE statement or he can specify arithmetic statements in which the result of computation is stored in such an item.

Since the main function of editing is to arrange data in a convenient format for reading, as for example in a report, an item described as containing editing symbols is a report item.

A report item, like any other item, is a storage area in memory, except the editing specified in the picture of the report item is performed on any data moved into the item. The following restrictions apply:

1. A report item can receive only numeric data; it is sometimes known as a numerically edited item.
2. A maximum of 18 numeric characters can be moved to a report item.
3. Editing clauses cannot be used with a computational-n item.
4. Editing clauses cannot be used with FILLER items.

The characters which may be used in a picture of a report item are as follows:

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

The characters 9 and V are discussed above. Their use in report items is exactly the same as in numeric items. The remainder are insertion and replacement characters.

The character $ may be replaced with a special currency sign whether it is used as an insertion or replacement character. The special currency sign must be defined in the SPECIAL-NAMES paragraph of the Environment Division. It is defined as a literal, limited to a single character which cannot be one of the following:

```
0 1 2 3 4 5 6 7 8 9
A B C D J K L P R S V X Z
* + - , ( ) .
```

Insertion Characters

When an insertion character is specified in the picture of a report item, it appears in the edited data item; therefore the size of the report item must reflect these additional characters. The insertion characters are as follows:

```
$ + - . , 0 B CR DB
```

$ When a single dollar sign is specified as the leftmost symbol in a report item picture, it appears as the leftmost character in the edited data item. This character is counted in the size of the report item. A special currency sign may replace the $ sign; it must be defined in the SPECIAL-NAMES paragraph.

\+ When a plus sign is specified as the first or last symbol of a report item picture, a display plus sign is inserted in the indicated character position of the edited data item, provided the data is positive (contains a positive operational sign) or is unsigned. If the data is negative, a minus sign is inserted in the indicated character position. This sign is counted in the size of the report item.

\- When a minus sign is specified as the first symbol or last symbol of a report item picture, a display minus sign is inserted in the indicated character position of the edited data item, provided the data is negative (contains a negative operational sign). If the data is not negative, a blank is inserted in the indicated character position. This sign or blank is counted in the size of the report item.

. (decimal point) This character is used in a report item picture to represent an actual decimal point as opposed to an assumed decimal point. When it is used, a decimal point appears in the edited data item as a character in the indicated character position. Therefore, the decimal point is counted in the size of the report item. A picture of a report item can never contain more than one decimal point, actual or assumed.

, (comma) When a comma is used in the picture of a report item, a comma is inserted in the corresponding character position of the edited data item. It is counted in the size of the report item.

0 When a zero is used in the picture of a report item, a zero is inserted in the corresponding character position in the edited data item. It is counted in the size of the report item.

B When this character is used in the picture of a report item, a blank is inserted in the corresponding character position in the edited data item. It is counted in the size of the report item.

CR This symbol, which represents credit, may be specified only at the right end of the picture of a report item. The symbol is inserted in the last two character positions of the edited data item, provided the value of the data is negative. If the data is positive or unsigned these last two character positions are set to blanks. Since this symbol always results in two characters, CR or blanks, it is included as two characters in the size of the report item.

DB This symbol, which represents debit, may be specified only at the right end of the picture of a report item. It has the same results as the credit symbol.

Examples of Insertion Characters:

Source data	Editing Picture	Edited Item
4 8	$99	$ 4 8
4 8 3 4	$99.99	$ 4 8 . 3 4
4 8 3 4	9,999	4 , 8 3 4
2 9 2	+999	+ 2 9 2
2 9 +2	+999	+ 2 9 2
2 9 -2	+999	- 2 9 2
2 9 -2	999-	2 9 2 -
2 9 +2	-999	Δ 2 9 2
2 9 2	999-	2 9 2 Δ
2 4 3 2 1	$BB999.99	$ Δ Δ 2 4 3 . 2 1
2 4 3 2 1	$00999.99	$ 0 0 2 4 3 . 2 1
1 1 3 -4	99.99CR	1 1 . 3 4 C R
1 1 3 4	99.99CR	1 1 . 3 4 Δ Δ
2 3 7 -6	99.99DB	2 3 . 7 6 D B
2 3 7 6	99.99DB	2 3 . 7 6 Δ Δ

Replacement Characters

A replacement character in the picture of a report item suppresses leading zeros in data and replaces them with other characters in the edited data item. The replacement characters are as follows:

Z * $ + -

Only one replacement character may be used in a picture, although Z or * may be used with any one of the insertion characters.

Z One Z character is specified at the left end of the report item picture for each leading zero that is to be suppressed and replaced by blanks in the edited data item. Z's may be preceded by one of the insertion characters $ + or - and interspersed with any of the insertion characters decimal point, comma, zero, or B.

Only the leading zeros that occupy a position specified by Z will be suppressed and replaced with blanks. No zeros will be suppressed to the right of the first non-zero digit whether a Z is present or not. Nor will any zeros to the right of an assumed or actual decimal point be suppressed unless the value of the data is zero and all the character positions in the item are described by a Z. In this special case, even an actual decimal point is suppressed and the edited item consists of all blanks.

If a $ + or - is present preceding the Z's, it is inserted in the far left character position of the item even if succeeding zeros in the item are suppressed. In the special case where the value of the data is zero and all the character positions following the $ + or - are specified by Z's, the $ + or - will be replaced by blanks.

If a comma, zero, or B in the picture of a report item is encountered before zero suppression terminates, the character is not inserted in the edited data item, but is suppressed and a blank inserted in its place.

* The asterisk causes the leading zeros it edits to be replaced by an asterisk instead of a blank. It is specified in the same way as the editing character, Z, and follows the same rules.

$ When the dollar sign is used as a replacement character to suppress leading zeros, it acts as a floating dollar sign and will be inserted directly preceding the first non-suppressed character. One more dollar sign must be specified than the number of zeros to be suppressed. This dollar sign will always be present in the edited data whether or not any zero suppression occurs. The remaining dollar signs act in the same way as Z's to effect the suppression of leading zeros. No other editing character may precede the initial dollar sign. Each dollar sign specified in a picture is counted in determining the size of the report item. A special currency sign may replace the $; it must be defined in the SPECIAL-NAMES paragraph.

\+ When a plus sign is used as a replacement character, it is a floating plus sign. The plus sign is specified one more time than the number of leading zeros to be suppressed. It functions in the same way as the floating dollar sign; a plus sign is placed directly preceding the first non-suppressed character if the edited data is positive or unsigned, and a minus sign is placed in this position if the edited data is negative.

\- When a minus sign is used as a replacement character, it is a floating minus sign. The minus sign is specified one more time than the number of leading zeros to be suppressed. It functions in the same way as the floating plus sign except that a blank is placed directly preceding the first non-suppressed character if the edited data is positive or unsigned.

Examples of Replacement Characters:

Source Data	Editing Picture	Edited Item
00923	ZZ999	ΔΔ923
00923	ZZZ99	ΔΔ923
000000	ZZZZ.ZZ	ΔΔΔΔΔΔΔ
00923	$***.99	$**9.23
00824	$$$9.99	ΔΔ$8.24
0052$\bar{6}$	---9.99	ΔΔ-5.26
3265	$$$.99	$32.65

Examples of Picture Editing:

Data to be Edited	Picture of Report Item	Edited Item
012345	ZZZ,999.99	Δ12,345.00
001234	Z99,999.99	Δ00,012.34
000123	$ZZZ,ZZ9.99	$ΔΔΔΔΔ1.23
000012	$ZZZ,ZZZ.99	$ΔΔΔΔΔΔΔ.12
001234	$***,**9.99	$**1,234.00
123456	$***,***.99	$123,456.00
123456	$***,***.99	$*****1.23
0000$\overset{+}{2}$	+999,999	+000,012
0000$\bar{2}$	-ZZZ,ZZZ	-ΔΔΔΔΔ12
12345$\bar{6}$	$ZZZ,ZZ9.99CR	$123,456.00CR
0012$\overset{+}{3}$	$ZZZ,ZZ9.99DB	$ΔΔΔΔΔ1.23ΔΔ
001234	$(4),$$9.99	ΔΔΔΔ$123.40
000000	$(4),$$$.99	ΔΔΔΔΔΔΔ$.00
0000$\bar{2}$	----,---.99	ΔΔΔΔΔΔΔ-.12
0000$\overset{+}{2}$	----,---.99	ΔΔΔΔΔΔΔΔ.12
000001	$$$$,$ZZ.99	illegal picture

Summary of Rules for Picture Editing Clauses

1. Only one of the characters of the set Z * $ + and - can be used within a single picture as a replacement character, though it may be specified more than once.

2. If one of the replacement characters Z or * is used with one of the insertion characters $ + or -, the plus or minus sign may be specified as either the leftmost or rightmost character in the picture.

3. A plus sign and a minus sign may not be included in the same picture.

4. A leftmost plus sign and a dollar sign may not be included in the same picture.

5. A leftmost minus sign and a dollar sign may not be included in the same picture.

6. The character 9 may not be specified to the left of a replacement character.

7. Symbols which may appear only once are: V, S, decimal point, CR, and DB.

8. The decimal point may not be the rightmost character in a picture.

POINT LOCATION IS $\left\{\begin{matrix}\text{LEFT}\\\text{RIGHT}\end{matrix}\right\}$ integer-2 [PLACES]

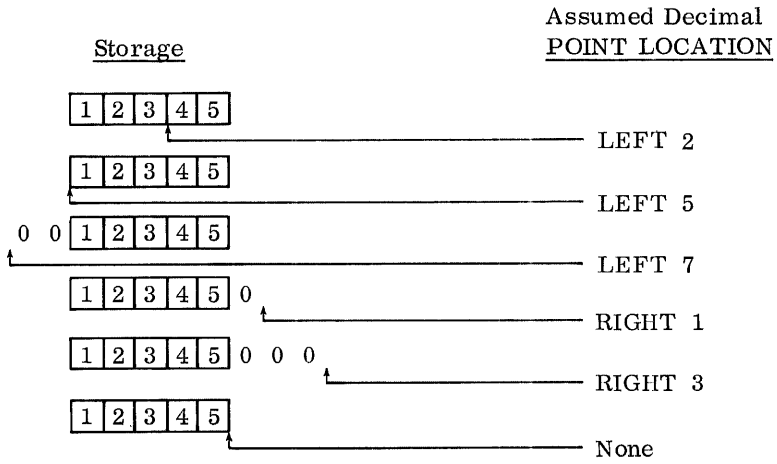
The POINT LOCATION clause specifies the position of an assumed decimal point in a numeric item (see PICTURE clause). An actual decimal point (one that occupies a character position in memory) cannot be specified by this clause; instead, a PICTURE clause must be used.

This clause can be used only in the description of an elementary item. When specified, the decimal point for the item is assumed to be integer-2 digit positions to the right (or left) of the rightmost digit in the item. Integer-2 is a numeric literal with positive integral value.

If a PICTURE clause is included in the entry, the POINT LOCATION clause is documentary only. If neither the PICTURE clause nor the POINT LOCATION clause is specified, the item is assumed to be an integer (no decimal point).

Example

If the digits 12345 are stored in an item of size 5, they are treated in accordance with the position of the assumed decimal point in that item, even though these digits are stored in exactly the same way in each case.



RECORD CONTAINS [ingeger-4 TO] integer-5 CHARACTERS[DEPENDING ON {RECORD-MARK / data-name-1}]

This clause specifies record size. If all the records in a file are the same size, integer-5 specifies the exact number of characters in each. When integer-5 only is used, the clause provides documentary information and has no effect on the object program.

When the records in a file are not all the same size, the RECORD CONTAINS clause or the OCCURS clause is used to specify record size. There are three ways to specify variable length records:

- RECORD CONTAINS clause with DEPENDING ON RECORD-MARK
- RECORD CONTAINS clause with DEPENDING ON data-name
- OCCURS clause with DEPENDING ON data-name

Only one type of variable length record may be used in a single file.

OCCURS is used to describe a file with records that have a fixed length base portion and a variable number of fixed length trailer portions.

RECORD CONTAINS with DEPENDING ON option is used to describe all other variable length records. Integer-4 and integer-5 mark the limits of record length in a particular file. Integer-4 is the number of characters contained in the smallest record in the file and integer-5 is the number of characters in the largest record.

DEPENDING ON RECORD-MARK is used when each record in a file is terminated by the special record-mark character]. For an input file, the RECORD-MARK character must be contained in the last item of each record in the file. To use this option with an output file, a single character item is defined with the value RECORD-MARK, and moved to the last character position in the record, or the statement, MOVE RECORD-MARK TO... may be used to place the record mark character in the last character position in the record.

DEPENDING ON data-name is used for a file in which each record contains a key item giving the record length in characters; data-name specifies this key item. It must be an elementary item, not exceeding 7 characters; it must appear as an entry in each record description; and occupy the same relative position in each record. Each key item must be the same in every respect including name, and the name must be unique to the whole DATA division.

A file described with this option may not be the subject of the RENAMING clause, nor may the COPY clause be used with any of the record descriptions.

$$\underline{\text{RECORDING}}\ \text{MODE IS}\ \left[\left\{\begin{matrix}\underline{\text{BINARY}}\\ \underline{\text{DECIMAL}}\end{matrix}\right\}\right]\ \left[\left\{\begin{matrix}\underline{\text{HIGH}}\\ \underline{\text{LOW}}\\ \underline{\text{HYPER}}\end{matrix}\right\}\ \text{DENSITY}\right]$$

The format phrase, the density phrase, or the entire clause may be omitted. When the clause is omitted, DECIMAL and HIGH are assumed. If format is omitted, DECIMAL is assumed; and if density is omitted, HIGH is assumed.

REDEFINES

The REDEFINES clause provides for specifying data in an alternate manner. It does not change the data, only the method of referencing it; this includes giving the item a new name. Both the original item and its redefinition occupy the same physical area in memory.

level-number data-name-4 REDEFINES data-name-5

This clause may be used at any level in a record description entry in the Constant, Working-Storage, or Common-Storage Sections. It may not be used at the 01 level in the File Section because each record named in the DATA RECORDS clause of a file description entry implies automatic redefinition of the record area. Independent items with level number 77 can be redefined, but items with level numbers 66 or 88 cannot be redefined. The level numbers and the sizes of data-name-4 and data-name-5 entries must be equal.

The data-name-4 entry must immediately follow the data-name-5 entry, or the last item within it if data-name-5 is a group item. Data-name-4 may be the name of a group or an elementary item regardless of the nature of the data-name-5 item. If it is a group item, data-name-4 is followed by the entries for all the items in the group. Redefinition ends when a level number less than or equal to that of data-name-4 is encountered. If data-name-4 is an elementary item it completely redefines the original area.

An OCCURS clause with the DEPENDING ON option cannot be specified for any entries in the original item or its redefinition. In addition, data-name-5 cannot contain any OCCURS clause nor be subordinate to an item containing an OCCURS. No value may be specified in the redefinition of an item.

When an area is redefined, all descriptions of the area remain in effect for the entire program. The one that is selected depends on the particular reference made to the area. For example, if two items A and B share the same area, MOVE X TO A moves X to the area according to the description of A and MOVE Y TO B moves Y to the same area according to the description of B. These statements could be executed anywhere in a program; the final contents of the area depends on the order in which they are executed.

An example of a use of the REDEFINES clause is given in the discussion of TABLES.

This clause permits alternate naming of elementary or group items not at the 01 level.

66-data-name-7 RENAMES data-name-8 [THRU data-name-9].

Data-name-7 may be used to refer to the item defined as data-name-8 or to the group of items from data-name-8 through data-name-9. One or more RENAMES entries can be written for any record description entry. All RENAMES associated with an entry must be written immediately following the last item description in that entry. The renamed items must be in the associated record description.

If the THRU option is included, data-name-7 is treated as a group item which includes all the elementary items beginning with data-name-8 or the first elementary item in data-name-8 if it is a group item, and concluding with data-name-9 or with the last elementary item in data-name-9 if it is a group item. When data-name-9 is specified, none of the elementary items within the range defined by THRU can be of variable length, nor may any item within the range have a lower level number than data-name-8. Data-name-8 and data-name-9 may not have the same name nor may data-name-9 be subordinate to data-name-8.

When the THRU option is not included, data-name-7 is treated as a group item if data-name-8 is a group item and as elementary if data-name-8 is elementary. Data-name-7 assumes the level number, class and usage of data-name-8.

A level 66 entry cannot rename another level 66 entry nor may it rename an independent level 77 entry, a condition name level 88 entry, or record level 01 entry. The REDEFINES clause may be used to provide alternate names for any 01 level entry not in the File Section. Data-name-7 cannot be a qualifier; data-name-8 and data-name-9 cannot be qualified or require qualification. Neither data-name-8 nor data-name-9 may contain an OCCURS clause or be subordinate to an item containing an OCCURS clause.

Examples:

1)
```
01 DATE-WORD.
     02 YEAR-1 PICTURE 99.
     66 YEAR-2 RENAMES YEAR-1.
     02 MONTH-1 PICTURE 99.
     66 MONTH-2 RENAMES MONTH-1.
     02 DAY-1 PICTURE 99.
     66 DAY-2 RENAMES DAY-1.
     66 DAY-3 RENAMES DAY-1.
```

2)
```
01 DETAIL.
     03 ITEM-NUMBER PICTURE 9(4).
     03 VENDOR-IDENT.
        05 VENDOR-CLASS PICTURE 9(3).
        05 VENDOR-NUMBR PICTURE 9(5).
     03 CUST-IDENT.
        05 CUST-CLASS  PICTURE 9(3).
        05 CUST-NUMBR  PICTURE 9(5).
     66 ALL-IDENT RENAMES VENDOR-IDENT THRU CUST-IDENT.
```

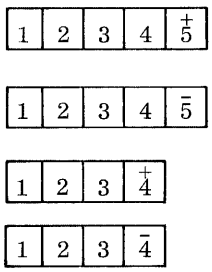
SIGNED

This clause indicates an operational plus or minus sign. If it is omitted, the item is assumed to be unsigned and positive. When this clause is specified, the processor assumes that an operational sign does not appear as a separate character on the external device, but is combined with the rightmost character in the item.

The SIGNED clause can be used only in describing elementary items, and an item specified as signed is assumed to be numeric; therefore, when SIGNED is included, CLASS may be omitted. If a PICTURE clause which specifies an operational sign is included in the entry, the SIGNED clause is not needed. SIGNED cannot be used in an entry which specifies editing since such an item must not have an operational sign. An operational sign, since it is not specified as a separate character on the external device, nor stored as a separate character in memory, is not counted in determining the size of the item.

Examples:

Computational Item in Storage



SIZE IS integer-1 [{CHARACTERS / DIGITS}]

The size of an elementary item is specified in terms of the number of character positions it occupies in memory. The size of every elementary item must be specified either in this clause or in a PICTURE unless the item is a floating point binary number (USAGE IS COMPUTATIONAL-2). SIZE for a group item is used for documentation only. If both SIZE and PICTURE clauses are specified, the size of the item is determined by PICTURE, and SIZE is used as documentation only.

The size of a numeric item is specified in DIGITS, and the size of an alphabetic or alphanumeric item in CHARACTERS; both words are optional in this context. The maximum size for an elementary item is 255 characters or digits. If the item is used in arithmetic calculations or is moved to an edited field, the maximum size is 18 digits. The maximum size of a numeric literal is 30 digits.

Operational signs and decimal points associated with numeric items do not occupy actual character positions in memory and are not counted in the size of an item. However, when the alphanumeric characters + or -, or the actual decimal point are stored as part of an edited item, they are counted in determining the size (PICTURE EDITING).

Examples:

```
SIZE IS 1 CHARACTER
SIZE IS 80 CHARACTERS
SIZE IS 5 DIGITS
SIZE IS 254.
```

```
SIZE IS integer-1 [ [ NUMERIC          ] ] [ [ COMPUTATIONAL-n ] ] [ [ CHARACTERS ] ]
                  [ [ ALPHABETIC       ] ] [ [ COMPUTATIONAL   ] ] [ [ DIGITS     ] ]
                  [ [ ALPHANUMERIC     ] ] [ [ DISPLAY         ] ]
                  [ [ AN               ] ]
```

Since CLASS and USAGE are both optional COBOL words, they can be omitted. If the SIZE clause is followed by class and usage clauses, and if the optional words are omitted from the last two clauses, the result is a single clause combining specifications of all three.

Examples:

Clause	Example in Storage
SIZE IS 12 ALPHABETIC DISPLAY CHARACTERS	E R R O R M E S S A G E
SIZE IS 3 NUMERIC COMPUTATIONAL DIGITS	1 2 3
SIZE IS 6 NUMERIC COMPUTATIONAL DIGITS	1 6 9 2 4 3
SIZE IS 9 ALPHANUMERIC DISPLAY CHARACTERS	S T O C K - 4 6 3
SIZE IS 7 AN DISPLAY CHARACTERS	$ * * 4 . 8 0
SIZE IS 8 COMPUTATIONAL DIGITS	0 0 0 0 1 2 3 4

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

The SYNCHRONIZED clause specifies space allocation for an elementary item transmitted by a move operation. Normally, for compatability with external devices, data items are packed without regard for machine words. This may make an item relatively inaccessible and require several instructions to reference the item, increasing running time. When an item is referenced often, it is sometimes preferable to specify exactly how it should be stored.

If SYNCHRONIZED LEFT is specified, the leftmost character of the item is allocated to the leftmost character position in the next available whole machine word; subsequent characters are allocated to successive character positions to the right. If the item occupies more than one machine word (ten character positions), succeeding characters are allocated to consecutive machine words, from left to right, as for the first word.

If SYNCHRONIZED RIGHT is specified, the item is allocated to as many of the next available whole machine words as are needed to contain it. The rightmost character of the item is allocated to the rightmost character position in the last word; preceding characters are allocated to successive character positions to the left in this word. If the item is allocated to more than one machine word, preceding characters are allocated to the preceding machine words, from right to left, as for the first word. The SYNCHRONIZED clause need not be used in the description of a COMPUTATIONAL-n item. All level 77 items are full word synchronized.

A group containing synchronized elementary items occupies more storage than is implied by the individual elementary items. If such a group is referenced (by a MOVE statement), the whole area occupied by the group is referenced, not just the portions containing the data. The receiving item must be large enough to contain the whole group not just the data. If synchronized elementary items are referenced individually, only the data is referenced, and the excess positions occupied by the item may or may not be ignored. No synchronization occurs when data is read or written.

When an entry contains both OCCURS and SYNCHRONIZED clauses, each occurrence of the item is synchronized. The initial contents of implied filler areas will not be guaranteed.

In the examples on the next page, shaded areas indicate unused portions of machine words.

Picture	Data	Item (machine words)	SYNCHRONIZED
S9(3)V	1 2 +3	0 0 0 0 0 0 0 1 2 +3	RIGHT
S9(3)V	1 2 −3	1 2 −3	LEFT
S9(3)V	1 2 −3	0 0 0 0 0 0 0 0 0 −1	RIGHT
S9(3)V	1 2 −3	0 0 −1	LEFT
S9(5)	1 2 −3	0 0 0 0 0 0 0 1 2 −3	RIGHT
S9(5)	1 2 −3	0 0 1 2 −3	LEFT

Picture	Data	Item (machine words)	JUSTIFIED	SYNCHRONIZED
X(9)	A B C D E F G H I	A B C D E F G H I	-	LEFT
X(9)	A B C D E F G H I	A B C D E F G H I	-	RIGHT
X(11)	A B C D E F G H I	A B C D E F G H I Δ Δ	-	LEFT
X(11)	A B C D E F G H I	Δ Δ A B C D E F G H I	RIGHT	RIGHT

Data is often organized in a table constructed as a group of constants. There are two methods for describing a table and for referencing items from a table. In the first method, each item is named and described individually, so that any item can be referenced by a name. In the second method, the table is redefined and the OCCURS clause is used to specify a shorthand description of the area in memory; an item in the table may then be referenced by subscripting instead of by name.

Consider a table that consists of the male and female populations of each of the 50 states for 1956, 1957, 1958, 1959, and 1960. These figures are arranged so that the male population of Alabama in 1956 is the first number in the table, followed by the female population of Alabama in 1956, and then by the corresponding pairs for Alabama in 1957, 1958, 1959, and 1960. This set of ten numbers is followed by the corresponding set of ten numbers for Alaska, Arizona, and so on, ending with the ten numbers for Wyoming. The 500 numbers in the table are organized according to alphabetic order of the states, then by order of year, then by sex, with the male number preceding the female number for each year within each state. Each population figure is an 8-digit number. This table could be described in the constant section of the Data Division as a constant record, as follows:

```
01 POP-RECORD
   02 ALABAMA
      03 FIRST-YEAR
         04 MALE PICTURE 99999999 VALUE IS xxxxxxxx.
         04 FEMALE PICTURE 99999999 VALUE IS xxxxxxxx.
      03 SECOND-YEAR
         04 MALE PICTURE 99999999 VALUE IS xxxxxxxx.
         04 FEMALE PICTURE 99999999 VALUE IS xxxxxxxx.
      03 THIRD-YEAR
         04 MALE PICTURE 99999999 VALUE IS xxxxxxxx.
         04 FEMALE PICTURE 99999999 VALUE IS xxxxxxxx.
      03 FOURTH-YEAR
         04 MALE PICTURE 99999999 VALUE IS xxxxxxxx.
         04 FEMALE PICTURE 99999999 VALUE IS xxxxxxxx.
      03 FIFTH-YEAR
         04 MALE PICTURE 99999999 VALUE IS xxxxxxxx.
         04 FEMALE PICTURE 99999999 VALUE IS xxxxxxxx.
   02 ALASKA
      03 FIRST-YEAR
         04 MALE PICTURE 99999999 VALUE IS xxxxxxxx.
         04 FEMALE PICTURE 99999999 VALUE IS xxxxxxxx.
      03 SECOND-YEAR
         04 MALE PICTURE 99999999 VALUE IS xxxxxxxx.
         04 FEMALE PICTURE 99999999 VALUE IS xxxxxxxx.
               .
               .
               .
```

During compilation, the processor associates the 500 values with particular areas in memory. At execution time, all of the numbers are stored in memory and are available for reference by the object program. The items are referenced by name. However, because of the manner in which the data names have been assigned, nearly all references to this table require qualification. For example, to refer to the female population of Arizona in 1959, the programmer specifies FEMALE OF FOURTH-YEAR OF ARIZONA. FIRST-YEAR OF ARKANSAS refers to the pair of population numbers for Arkansas in 1956. OHIO is a reference to the ten population numbers for that state. To refer to all 500 numbers in the table, the programmer simple specifies the name of the record, that is, POP-RECORD.

This is the first method of describing and referencing a table. A shorthand method for describing this table and permitting subscript references is possible using the REDEFINES and OCCURS clauses.

A particular item or set of items can be referenced by its position in the table rather than by its name. For example, the female population of Alabama in 1960 is the second number in the fifth pair of numbers in the first set of numbers (referenced by name as FEMALE OF FIFTH-YEAR OF ALABAMA). The constant area consists of 50 similar items, each called POPULATION; and each item consists of 5 similar items, each called YEAR. Each item called YEAR consists of 2 similar items, each called MALE-FEMALE and each having a picture of 99999999.

To refer to this table with subscript numbers, the constant area must be redefined, by means of the REDEFINES clause, as a table using the OCCURS clause. The new table is specified as follows:

```
01  POP-TABLE REDEFINES POP-RECORD.
    02  POPULATION OCCURS 50 TIMES.
        03 YEAR OCCURS 5 TIMES
           04 MALE-FEMALE PICTURE 99999999 OCCURS 2 TIMES.
```

Redefinition of an area does not change any information in that area; it simply provides a different way for the object program to interpret that area.

A table of binary numbers can be constructed, but an OCCURS clause cannot be specified for a computational-n item; therefore, a dummy group name must be set up for a table of binary numbers.

```
01  TABLE
    02  GROUPF OCCURS n TIMES
        03 FI SIZE 8 COMPUTATIONAL-1 CHARACTERS
```

Computational (BCD) and computational-n (binary) items may not be included in the same table.

SUBSCRIPT NUMBERS

Any item described by OCCURS can be referenced by subscript numbers. Subscript numbers are specified in parentheses following the name of the type of item required. They are written in descending order of inclusiveness and are separated from each other by a comma and a space.

A single item in the table, POP-TABLE, is referenced by three subscript numbers; one to indicate the POPULATION item, one to indicate the YEAR item, and one to indicate which of the two items within the chosen YEAR is required. For example, the male population of Alaska in 1960 is referenced by MALE-FEMALE (2,5,1). A pair of items is referenced by two subscript numbers; one to indicate in which particular POPULATION item the required pair occurs, and one to indicate which YEAR item is required. The pair of numbers for Wyoming in 1957 is referenced by YEAR (50,2). Similarly, a set of five pairs of numbers is referenced by just one subscript number; the ten numbers for Arizona are referenced by POPULATION (3). Similarly, all 500 numbers are referenced by the name of the table, with no subscripting numbers, that is, by POP-TABLE. This second method of describing and referencing a table requires that the table be described in the longhand manner as well as in the shorthand manner.

Tables to be handled by COBOL must not require more than three subscript numbers to refer to any particular item. Any table can require one, two, or three subscript numbers, and any reference to an item in such a table is said to require one, two, or three levels of subscripting.

It is not necessary to state the subscript directly in numeric form. The subscript specified in parentheses can be the name of a data item in the program. When the table reference is executed in the object program, the current value of the data item used as a subscript is used to calculate the table item required. The data-name used as a subscript may not be qualified, although qualified items can be subscripted. A data-name defined as binary and a data-name defined as decimal may both be used as subscripts. Subscripts may be positively signed and have implied point locations but they must be integral in value. Literals must be positive integers; they are converted to binary at compile time.

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

Although this clause specifies that an item will be used primarily for computation or display; it does not necessarily limit it to that usage. Computational-n items may not be used for display. If usage is not specified it is assumed to be display. The USAGE clause can describe an item at any level. At the group level it is inclusive and cannot be contradicted by any item in the group. USAGE is an optional COBOL word; it is omitted when SIZE, CLASS and USAGE are combined, (see SIZE-CLASS-USAGE).

DISPLAY describes items involved in operations that edit or otherwise prepare data for reading; they may be alphabetic, alphanumeric, or numeric. A numeric display item may be used for computations as well as display, and COMPUTATIONAL items may be used for display.

A COMPUTATIONAL item is a decimal numeric value. It is packed automatically, although the user may control placement with the SYNCHRONIZED and JUSTIFIED clauses. This usage provides efficient storage but requires conversion to binary during multiplication, division and exponentiation. Addition and subtraction is done in display arithmetic and no conversion is required.

Both computational and computational-n items must be numeric and size may not exceed 18 decimal digits.

COMPUTATIONAL-n items may be COMPUTATIONAL-1 or COMPUTATIONAL-2, and are stored as binary numbers. When Common-Storage, Working-Storage, or Constant items are defined as COMPUTATIONAL-n (binary) and contain an initial value, this value is converted to binary. Computational-n items may not be used for display; nor may they be synchronized left or used to describe an item that is to be edited.

COMPUTATIONAL-1 items are stored as un-normalized, floating, single precision integers for 14 or less digits; and stored as normalized, floating, double precision integers for 15-18 digits.

COMPUTATIONAL-2 items are stored as floating-point binary numbers, in single precision, occupying one computer word. The decimal point is carried within the item as a binary exponent. Size is immaterial for such an item, and a PICTURE or equivalent clause has no meaning.

The computational-n option reduces the number of conversions in an arithmetic statement, as all internal arithmetic operations are performed in the binary mode. Double precision arithmetic may be avoided by specifying computational-2 or by reducing the size of a computational-1 item to less than 15 decimal digits.

Examples:

Picture	Item in Memory
SIZE IS 18 COMPUTATIONAL-2 DIGITS VALUE IS 123.	1726 | 7540 | 0000 | 0000 | 0000
SIZE IS 18 COMPUTATIONAL-1 DIGITS VALUE IS 123.	1726 | 7540 | 0000 | 0000 | 0000 1646 | 0000 | 0000 | 0000 | 0000
SIZE IS 18 COMPUTATIONAL DIGITS VALUE IS 123.	␣ ␣ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3
SIZE IS 13 COMPUTATIONAL-1 DIGITS VALUE IS 123.	2000 | 0000 | 0000 | 0000 | 0173

In the above examples $123_{(10)} = 173_{(8)}$

See Appendix I for a full explanation of the 6000 Series floating point format as it pertains to COBOL computational usage.

The VALUE clause sets initial values and specifies values associated with condition names.

Format 1:

VALUE IS literal-1

Format 2:

$$\left\{\begin{matrix}\text{VALUE IS} \\ \text{VALUES ARE}\end{matrix}\right\} \text{literal-2 [THRU literal-3] [literal-4 [THRU literal-5]]...}$$

The first format is always used to define the value of a constant-item, or the initial contents of a working-storage or common-storage item. VALUE must be specified for each item in the Constant Section. In the Common-Storage or Working-Storage Sections, VALUE may be used to define initial values or values associated with a condition name. Initial values are set at the start of program execution; they are not affected by editing specifications.

The second format defines the value or range of values associated with a condition name. When a single value is associated with a condition name, only literal-2 is specified. When condition names represent more than one value or a range of values, as many THRU options may be used as needed. In the File Section, VALUE may be used only to define values associated with a condition name. It is ignored in any attempt to define initial values.

In the Report Section, VALUE is used in an elementary item description; it acts as a constant and is not changed during execution. The specified value is output each time the associated report group is output, (Chapter 5, SOURCE-SUM-VALUE).

If a VALUE clause is used at a group level, it may not appear in any entry for an item within that group; nor may a VALUE clause appear in an entry which contains an OCCURS clause or which is subordinate to one containing an OCCURS.

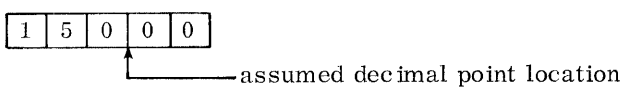
Literal-1 may be a literal or figurative constant of the same class as the item assigned a value. The literals in Format 2 must be the same class as the item with which the condition is associated. The maximum size of a non-numeric literal is 255 characters or digits; the maximum size of a numeric literal is 30 digits. Any digits over the number specified as the field size must be zeros required to locate the decimal point.

Examples:

1) The value of an independent constant item is defined:

   77 FICA-MAX SIZE IS 5 NUMERIC DIGITS POINT LOCATION IS LEFT 2 VALUE IS 150.00.

   The constant item will appear as follows synchronized within a word of storage:



2) Value is specified for the range of values of a condition name associated with an item:

```
01  YEARS PICTURE IS 9(4)
    88  SIX-YEARS VALUES ARE 1951 THRU 1956.
```

3) Value is specified for the initial value of an item:

```
01  HEADING-A.
    03  FIRST-WORD PICTURE X(10) VALUE IS "COBOL-LIST".
```

PROCEDURE DIVISION 4

Statements in the Procedure Division describe the operations to be performed by the object program. These statements are combined to form sentences and paragraphs; paragraphs may be combined to form sections. Paragraphs and sections are both referred to as procedures, and paragraph names and section names are both called procedure names. The elements of the statements are COBOL words, user-defined names, and literals. A summary description of these elements and of procedure-names is contained in Appendix A.

The Procedure Division may also contain Declaratives (see 4.2).

Execution of the object program begins with the first statement of the Procedure Division, excluding Declaratives. Statements are then executed in order of appearance except where the sequence is altered according to a rule specified in this chapter.

4.1 SPECIFICATION OF PROCEDURE DIVISION

```
PROCEDURE DIVISION.
[ DECLARATIVES.
  section-name-1 SECTION.
  introductory-sentence-1.
  paragraph-name-1
      .
      .
      .
  END DECLARATIVES. ]
[section-name-2 SECTION [priority-number]].
paragraph-name-2.
      .
      .
      .
paragraph-name-3.
[section-name-n SECTION [priority-number]].
 paragraph-name-n.
```

The division begins with the header on the first line. When declaratives are included, the entire declarative portion of the specification is written immediately following the division header. If declaratives are not included, the next line is the section name followed by a space and the word SECTION.

The paragraph names and texts are then specified for all paragraphs in the first section; each paragraph name is written on a new line. After the last sentence of the first section, the second section header is specified on a new line, followed by the paragraph names and texts, and so on for the entire division.

All paragraphs following a section header are assumed to be part of that section. Paragraphs that are not part of sections must be written at the beginning of the division, preceding the first section header. The user may specify that the program contains no sections by omitting section headers and writing the paragraphs consecutively.

A priority number following the word SECTION in the header indicates that the division is segmented. When segmentation is used, all the paragraphs in the division must be contained in sections, priority numbers may be specified for the sections (Segmentation 4.3).

The division header and all section and paragraph headers begin in column 8 and are terminated by a period, the remainder of each header line must be blank unless it is a paragraph name. The text of a paragraph may begin on the same line as the paragraph name, separated from it by at least one space, or it may begin on a subsequent line starting in column 12. All lines in the text of a paragraph begin in or to the right of column 12. The rules for splitting a word or literal between two lines are given in Appendix A.

4.2 DECLARATIVES

Declaratives define procedures to be executed in addition to standard error and label record handling procedures of the control system. A declarative consists of an introductory sentence containing one of the statements USE or ENTRY and the associated procedures. The introductory sentence defines the conditions under which these procedures should be executed, and is specified according to a fixed format. Procedures specified in a declarative are executed automatically under the input-output control system, according to the conditions specified in the USE sentence.

Each declarative is specified in a section by itself; it is preceded by a section header. The introductory sentence follows the header. The procedures are specified according to the same rules as all other procedures in the program. All declarative sections are grouped at the beginning of the Procedure Division under the collective header DECLARATIVES. They are followed by the collective termination header END DECLARATIVES.

Declarative procedures may reference or be referenced by procedures in the main body of the Procedure Division or in another declarative section. These references may be made only by the PERFORM statement. A referenced procedure in the main program must not contain a call to a declaratives procedure, as this could cause the return of control to or from the main program to be destroyed.

Input-output verbs must not be used in a declaratives section except that a non-standard tape output file may be closed in a USE BEFORE ENDING file label procedure.

4.3 SEGMENTATION

At execution time, sections of the Procedure Division may be separated into overlayable segments. Each segment becomes a separate subroutine in the relocatable binary output from compilation. The user indicates the overlay requirements with priority numbers following each section header, and the compiler provides the necessary controls.

The Declarative portion of the Procedure Division may not be segmented, nor may the Identification, Environment, and Data Divisions be segmented.

Segmentation does not affect the logical sequence of the program as specified in the source program. If any reordering of the object program is required to handle the flow from segment to segment, the compiler provides the control transfers to maintain the logic flow specified in the source program. An automatic jump is made when the succeeding section is in a different segment. Control may be transferred to any paragraph in a section, and it is not mandatory to transfer control to the beginning of a section. Any reference to another segment causes the referenced segment to be accessed from a storage device and loaded into memory with attendant delay in execution. The number of transfers should, therefore, be kept to a minimum.

Priority Numbers

Segmentation is accomplished through a system of priority numbers. The priority number is included in the section header. When segmentation is used, the entire Procedure Division will be in sections; each section is classified as belonging to the fixed or main segment or to one of the overlayable segments of the object program. The format for the section header is:

```
section-name SECTION [priority-number].
```

The section name is formed exactly as any procedure name; the priority number is 1-99. If the priority number is 1-49, or omitted, the section is part of the fixed or main segment. Overlayable segments are specified by priority numbers 50-99. In the above 50-99 range, all sections having the same priority number are combined into one segment. Each time a GO TO or PERFORM references a procedure in an overlay segment containing a group of sections with the same priority number, the entire segment is loaded.

The fixed or main segment is kept in memory at all times. It is, therefore preferable to give any frequently referenced sections priority numbers in the range 1-49. The overlayable segments of any subprogram must be compiled at the same time.

If a SORT is part of an overlayable segment, the input-output procedures associated with it must be in the same overlayable segment (sections with the same priority number) as the SORT statement, or they must be in the main segment. If the SORT statement is in the main segment, the associated input and output procedures must be in the main segment. The input and output procedures may not transfer control to another segment.

Example:

```
PROCEDURE DIVISION.
START-PROGRAM SECTION 1.
    .
    .
    .
READ-INPUT SECTION 2.
    START.
        .
        .
    STEP-2.
        .
        .
INPUT-TO-SORT SECTION 6.
    .
    .
    .
SORT-MASTER SECTION 50.
    .
    .
    .
OUTPUT-FROM-SORT SECTION 7.
    .
    .
    .
UPDATE-MASTER SECTION 51.
    .
    .
    .
PRINT-REPORT SECTION 8.
    .
    .
    .
```

4.4 STATEMENTS AND SENTENCES

The Procedure Division may contain imperative, conditional, and processor-directing statements. Each must be part of a sentence. Sentences must be terminated by a period and may contain a separator: the word THEN, or the semi-colon. A separator must not be followed by another separator. Separators may be used between statements, or in an IF statement.

Imperative Statements

These statements indicate operations for the object program to perform. They may consist of a series of statements separated by a space or a separator. The imperative verbs are:

ACCEPT ADD† ALTER CLOSE COMPUTE† DISPLAY DIVIDE† EXIT GO TO
GENERATE†† INITIATE†† MOVE MULTIPLY† OPEN PERFORM RELEASE
SEEK SORT STOP SUBTRACT† TERMINATE†† USE WRITE†††

† Without the SIZE ERROR option
†† Report Writer only
††† Without the INVALID KEY option

Conditional Statements

A conditional statement specifies that a condition is to be evaluated for truth and subsequent action of the object program is dependent on this truth value. A conditional statement may be preceded by an imperative statement in the same sentence. The conditional verbs are:

IF READ RETURN arithmetic statements with ON SIZE ERROR

WRITE with INVALID KEY

Processor-Directing Statements

These statements indicate operations for the processor to perform at compilation time. The processor-directing verbs are:

ENTER INCLUDE NOTE

4.5 CONDITIONS

Conditional expressions specify the conditions under which operations are to be performed or by-passed. In simple conditional expressions, truth or falsity depends upon one condition only. A compound conditional expression is two or more simple conditional expressions connected by the logical operators AND and OR.

4.5.1 SIMPLE RELATIONAL CONDITION

The five simple conditions are:

relational

sign

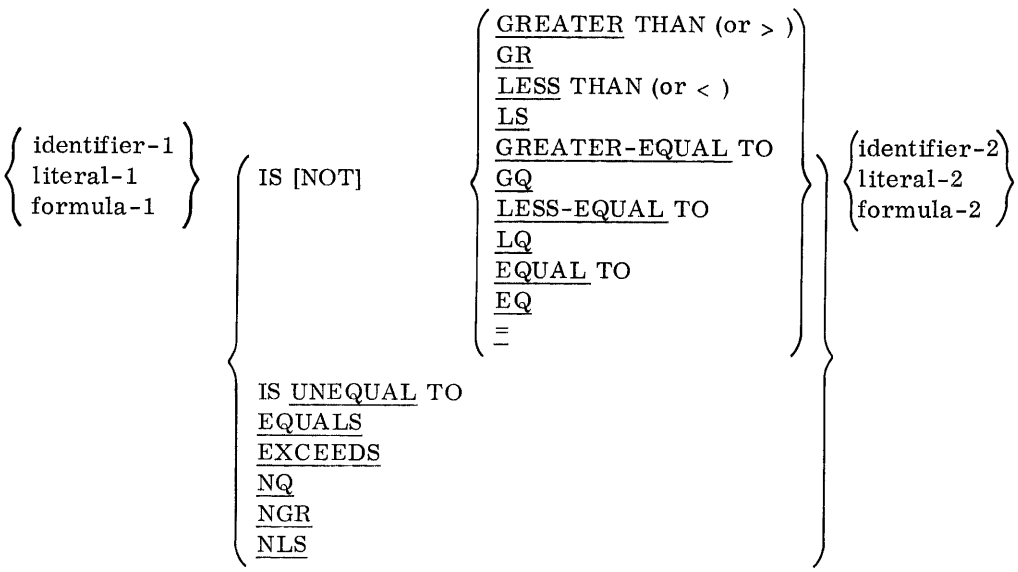
class

condition-name

switch-status-name

RELATIONAL

Using this method, the truth or falsity of the expression depends upon the relative magnitudes of two operands. Comparison of the operands is made by relational operators.



When a simple relational condition is specified, a comparison is made between two operands according to the relational operator. The operands may be literals, named data items, or formulas. Each is referred to as "item" in the following discussion. The manner in which two values are compared depends on whether they are numeric or non-numeric.

NUMERIC ITEMS

When both items are BCD numeric, the comparison is based directly on their algebraic values. Any difference in item length is ignored; 076000 is considered to be equal to 00000760, provided assumed decimal points in the two items correspond. If no assumed decimal point exists, each item is an integer and the first is considered 76,000 and the second 760.

Binary to binary comparison — a binary number can be compared to another binary number provided they are data-names specified COMPUTATIONAL-n.

Binary to literal comparison — the literal is converted to binary at compile time and a binary to binary test is made at execution time.

Binary to decimal comparison — the decimal is converted to binary at execution time and a binary to binary test is made.

A COMPUTATIONAL-n field can be used in POSITIVE, NEGATIVE, and ZERO tests. Since zero is considered a unique value, neither positive nor negative items with zero values are considered to be equal, regardless of length, decimal points, and sign.

NON-NUMERIC ITEMS

Two non-numeric, or one non-numeric and one numeric items are compared according to the COBOL collating sequence (Appendix B). The sequence is an ordering of the COBOL character set; each character has a specific place or position in the set.

Items of equal length are compared character by character as specified by the relational operator. Comparison begins at the left and terminates when inequality is encountered or the end of the items is reached, whichever occurs first. If no unequal pairs are detected, the items are equal. Unequal characters are compared according to their relative positions in the collating sequence. The item which contains the higher character in the collating sequence is considered to be the greater.

If items are of unequal length, comparison proceeds as above, with the following difference. If unequal characters are not detected before the end of the shorter item, the longer item is considered to be greater. If, however, the remaining characters are all blanks (or spaces); the items are considered to be equal.

SIGN CONDITION

The user may test whether the value of an item or formula is positive, negative, or equal to zero. CLASS must be numeric, and for the positive or negative test, the item must be signed. The value zero, whether signed or not, is considered to be neither positive nor negative.

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

CLASS CONDITION

This expression is used when the condition is to be based on the class of an item (CLASS clause, Chapter 3). A numeric or alphanumeric item may be tested for being NUMERIC, or an alphabetic or alphanumeric item for being ALPHABETIC.

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

CONDITION-NAME CONDITION

Only the condition-name associated with the required value of an item is specified.

[NOT] condition-name

This expression is used to test whether the value of a conditional variable is equal to the value(s) associated with the condition name. Condition name entries require an 88 level number identification and a VALUE clause. A condition name may be qualified by the elementary condition variable with which it is associated and the qualifiers of the condition variable.

SWITCH-STATUS-NAME CONDITION

Only the switch-status-name associated with the switch setting is required.

[NOT] switch-status-name

This expression is used when the condition is based on the status of a machine switch (SWITCH clause, Chapter 2).

4.5.2 COMPOUND CONDITIONAL EXPRESSIONS

Any two simple conditional expressions, specified according to any of the five methods described above, may be connected by a logical operator, AND or OR, to form a compound conditional expression. Further simple conditional expressions may be connected to this compound expression by logical operators to form complex conditional expressions.

AND Both expressions must be true if the whole expression is to be true. A AND B is true when both conditional expressions A and B are true.

OR One or both expressions must be true if the whole expression is to be true. A OR B is true when A is true, or when B is true, or when both A and B are true.

Parentheses indicate the order in which conditions in an expression are to be evaluated. Parentheses must always be paired; evaluation begins with the innermost pair and proceeds to the outermost pair. When the order of evaluation is not specified by parentheses, the expression is evaluated according to the logical operators AND and OR as follows: beginning at the left of the entire expression, each AND expression is evaluated and then each OR expression is evaluated.

For example, the conditional expression A AND B OR C AND D is considered as (A AND B) OR (C AND D). (A AND B) is evaluated first, followed by (C AND D). The whole expression is true if either A AND B is true (both A and B must be true) or if C AND D is true (both C and D must be true) or if both A AND B and C AND D are true.

The word NOT may be used in two ways. In simple conditional expressions, it indicates an opposite condition. For example, A IS NOT LESS THAN B, is the opposite of A IS LESS THAN B. The word NOT may also precede a conditional expression to indicate that when the expression is evaluated, the opposite condition is desired, as in NOT EXEMPT, where EXEMPT is a condition name. Similarly, NOT may precede the left parenthesis indicating that the opposite of the compound conditions is desired. NOT (A EQUALS B OR A EQUALS C) means that the condition is true only when A does not equal both B and C. NOT (A EQUALS B AND A EQUALS C) means that the condition is true when A does not equal either B or C.

The following table illustrates the rules for legal symbol pairs in compound conditional expressions. The letter C indicates a conditional expression. The letter P indicates that the specified pair is permissible, and the dash indicates that they are not.

		Second Symbol					
		C	OR	AND	NOT	(	)
First Symbol	C	-	P	P	-	-	P
	OR	P	-	-	P	P	-
	AND	P	-	-	P	P	-
	NOT	P	-	-	-	P	-
	(	P	-	-	P	P	-
	)	-	P	P	-	-	P

4.5.3 IMPLIED ELEMENTS

The data-name, literal, or formula to the left of the relational operator is the subject of the conditional expression, and the item to the right is the object. If a single conditional sentence contains a sequence of fully-stated simple conditional expressions (either consecutively, as in a compound conditional expression, or separately by verbs, key words, etc., some elements may be omitted without altering the meaning. The same subject and/or the same object, and/or the same relational operator must be specified for each expression. All but the first occurrence of such common elements may be omitted within the limitations explained below. Elements omitted in this way are implied elements.

Three types of omissions can be made: subject only is omitted, both subject and relational operator are omitted, or both subject and object are omitted. The processor derives the missing elements from the last preceding conditional expression in which subject, relation, and object are explicitly stated. Therefore, all but the first of any common elements in a sentence may be omitted. The presence or absence of parentheses in the unabbreviated form does not affect the use of abbreviated forms.

IMPLIED SUBJECT

The conditional expressions in a sentence have a common subject:

IF A EQUALS B OR A IS LESS THAN C

can be abbreviated to

IF A EQUALS B OR IS LESS THAN C

IF A EQUALS B MOVE M TO N OTHERWISE IF A IS LESS THAN C ADD X TO Y

can be abbreviated to

IF A EQUALS B MOVE M TO N OTHERWISE IF LESS THAN C ADD X TO Y

IMPLIED SUBJECT and RELATION

The conditional expressions in a sentence have a common subject and a common relation:

IF A EQUALS B OR A EQUALS C

can be abbreviated to

IF A EQUALS B OR C

IF A EQUALS B ADD X TO Y OTHERWISE IF A EQUALS C AND A EQUALS D MOVE M TO N

can be abbreviated to

IF A EQUALS B ADD X TO Y OTHERWISE IF C AND D MOVE M TO N

IMPLIED SUBJECT and OBJECT

The conditional expressions in a sentence have a common subject and a common object:

IF A EQUALS B OR A IS GREATER THAN B MOVE C TO A IF A IS GREATER THAN B ADD B TO A

can be abbreviated to

IF A EQUALS B OR IS GREATER MOVE C TO A IF GREATER ADD B TO A

In the unabbreviated form, A is the subject in the EQUALS and the two GREATER THAN conditional expressions, and B is the object in all three expressions. After their first explicit occurrence, both A and B may be implied. The value of A in the second GREATER THAN conditional expression is C, since C is moved to A by the previous statement. This sentence also illustrates the concept of nested conditionals.

IMPLIED LOGICAL OPERATOR

A logical operator may be implied when subjects, relations, and logical operators are common to all the conditional expressions. Only the first occurrence of subject and relation is stated explicitly; all objects are stated explicitly in a series (commas are optional) following the relation; the last object is preceded by an explicit specification of the common logical operator.

IF X EQUALS 2 OR X EQUALS Y OR X EQUALS Z MOVE M TO N

can be abbreviated to

IF X EQUALS 2, Y, OR Z MOVE M TO N

If the subject, relation, and logical operator are implied; none of the objects may be formula.

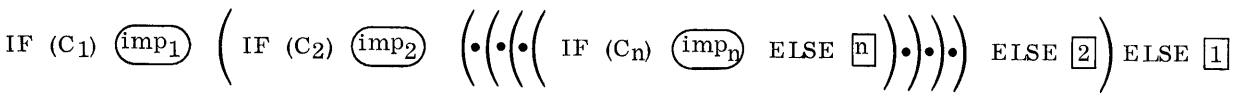
4.5.4 NESTED CONDITIONAL STATEMENTS

The general form of the conditional statement is as follows:

IF conditional-expression

$$\left\{\begin{array}{l}\text{statement-1}\\ \underline{\text{NEXT SENTENCE}}\end{array}\right\}\ \left\{\begin{array}{l}\underline{\text{OTHERWISE}}\\ \underline{\text{ELSE}}\end{array}\right\}\ \left\{\begin{array}{l}\text{statement-2}\\ \underline{\text{NEXT SENTENCE}}\end{array}\right\}$$

Each of statement-1 and statement-2 can be an imperative statement, a series of imperative statements, or imperative statements followed by a conditional statement. If either or both statement-1 and statement-2 contain a conditional statement, then the conditional statement is said to be nested. A nested conditional statement may also contain conditional statements in the same way, and these too are said to be nested. Conditional statements may be nested to depth of 25 levels. The fullest form of nested conditional statements may be represented as follows:



Each (C_i) represents a conditional expression, each (imp$_i$) represents an imperative statement or a series of imperative statements, and each box represents an imperative statement or a conditional statement (which itself may contain conditional statements, and so on) or NEXT SENTENCE. (C_n) represents the conditional expression in the first conditional statement that does not itself contain any further conditional statements.

In a conditional statement, the word ELSE (or OTHERWISE) and a statement is specified once for each specification of the word IF and a conditional expression. The number of occurrences of the word ELSE (or OTHERWISE) in any conditional statement, however complicated by nestings, must be equal to the number of occurrences of the word IF, with the following exception.

If the phrase ELSE (or OTHERWISE) NEXT SENTENCE directly precedes the terminal period of a sentence, the entire phrase may be omitted and the period specified at the end of the previous phrase. The same rule may be applied to the resulting sentence, and so on.

In this discussion, IF represents the word IF and its associated conditional expression and ELSE represents the word ELSE (or OTHERWISE) and its associated statement. For each ELSE, the associated statement is executed only when the conditional expression in the corresponding IF is found to be false. The numbers specified in the boxes and the number indicated with the conditional expressions show the association between the two elements in a conditional statement. For example, the statement represented by [j] is executed only when the conditional expression (C_j) is false.

The processor scans the conditional statement for the first occurrence of ELSE and associates this with the first preceding IF. It associates the second occurrence of ELSE with the next preceding IF not previously associated with another ELSE, and so on for all the IF and ELSE pairs in the statement. If there are more occurrences of IF than ELSE in the statement, the processor infers that the programmer omitted that number of ELSE NEXT SENTENCE phrases at the end of the sentence. The curved lines in the illustration indicate the IF and ELSE pairs.

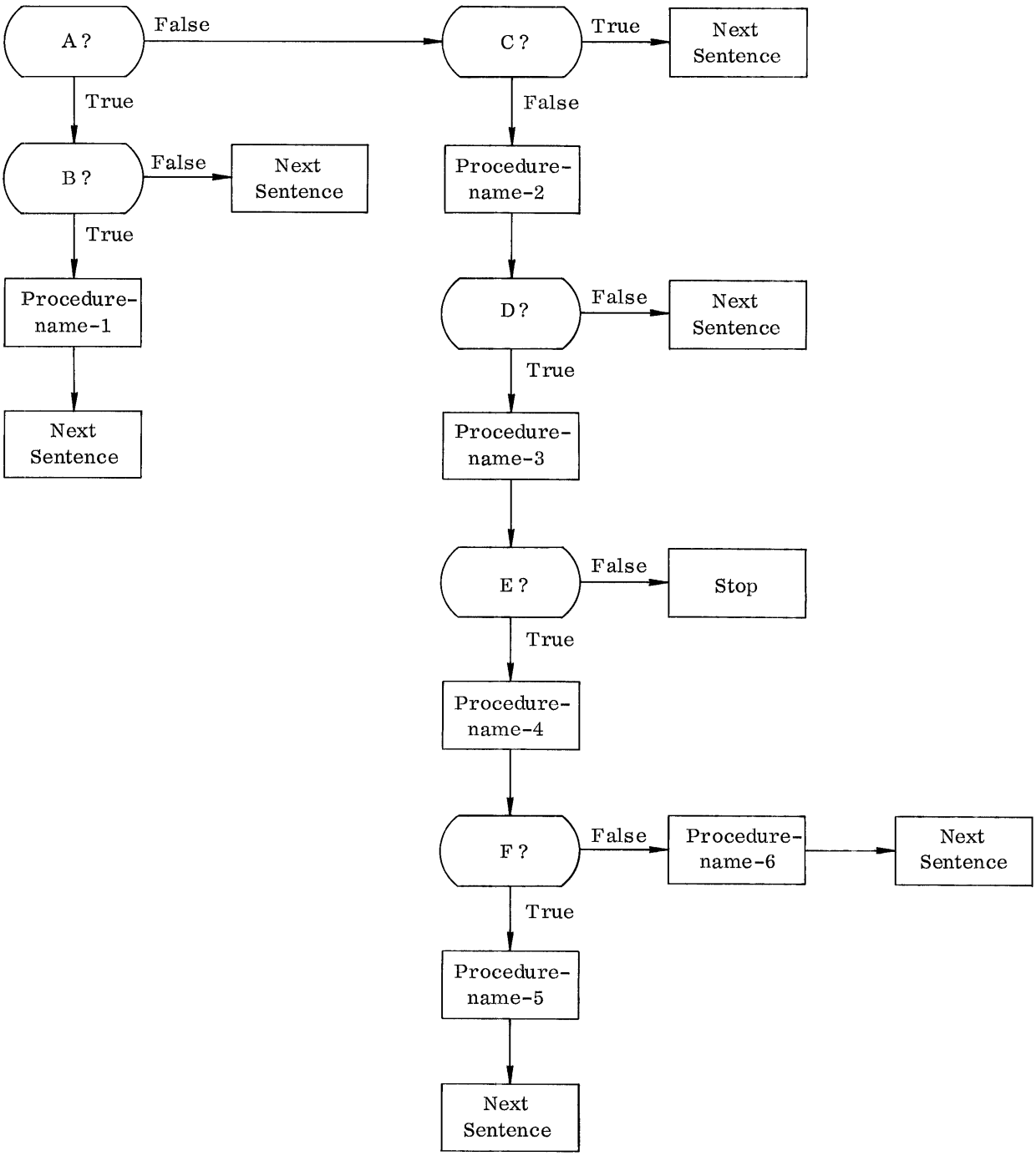
When the conditional statement is executed, (C_1) is evaluated first. If it is true, (imp_1) is executed, if this is specified, and (C_2) is evaluated. If this is true, (imp_2) is executed, if this is specified, and (C_3) is evaluated; and so on until (C_n) is reached. If this is true, (imp_n) is executed and control passes to the next sentence. No (imp_i) save for (imp_n) should specify a GO TO statement or NEXT SENTENCE, otherwise no subsequent conditional expressions can be evaluated. (imp_n) can be any type of imperative statement or NEXT SENTENCE, since no conditional expressions follow it.

If one of the conditional expressions (C_i) is found to be false, the corresponding (imp_i) is not executed nor are any subsequent conditional expressions in that nest evaluated; instead, the statement associated with the corresponding ELSE is executed. The remainder of the sentence is executed normally, unless the first word following the statement is ELSE, in which case, control passes directly to the next sentence. A sentence can contain more than one independent conditional statement, each of which can contain nested conditional statements.

The following sentence contains two independent nests of conditional statements. The first nest ends after the statement PERFORM procedure-name-2; the second nest consists of the remainder of the sentence and has an implied ELSE NEXT SENTENCE before the period. A, B, C, D, E, F each corresponds to a conditional expression.

```
IF A THEN IF B PERFORM procedure-name-1 ELSE NEXT SENTENCE ELSE IF C NEXT
SENTENCE ELSE PERFORM procedure-name-2 IF D PERFORM procedure-name-3 IF E
PERFORM procedure-name-4 IF F PERFORM procedure-name-5 ELSE PERFORM procedure-
name-6 ELSE STOP RUN.
```

The following flow-chart indicates the execution of this sentence.



4.6 ARITHMETIC EXPRESSIONS AND STATEMENTS

EXPRESSIONS

An arithmetic expression is a combination of data names, numeric literals, and the figurative constant ZERO(S) (ES) connected by arithmetic operators such that the expression reduces to a single value when it is evaluated at program execution time. An arithmetic expression may be used in the COMPUTE statement or as a subject or object in a conditional statement. The operators are as follows:

+	Addition	*	Multiplication
-	Subtraction	/	Division
**	Exponentiation		

The manner in which an arithmetic expression is to be evaluated may be specified by paired parentheses. Evaluation begins with the innermost pair and proceeds to the outermost pair. The terms within the parentheses are evaluated according to the order of precedence specified below.

A + or - used as a sign is considered first; then exponentiation is performed, then multiplication and division. Addition and subtraction are performed last. Evaluation begins at the left of each term and moves to the right. Thus, expressions which appear to be ambiguous algebraically, are permitted in COBOL. For example, A - B * C is interpreted by COBOL as A - (B * C). When no parentheses are specified, the whole of the arithmetic expression is evaluated according to the rules just stated. The processor assumes parentheses enclose the pairs of items on either side of multiplication and division signs, and the pairs of items on either side of addition and subtraction signs.

For example, the expression A / B - C * D ** H + E + F / G is interpreted by the processor as (A / B) - (C * (D ** H)) + E + (F / G). Spaces must appear on either side of any operator included in an arithmetic expression.

The rules for specifying the operators, parentheses, and a variable (date name, literal, figurative constant), are given in the following table. P indicates that a specified pair of symbols is permitted; a dash indicates that it is not.

		Second Symbol					
		Variable	* or /	- or +	**	(	)
First Symbol	Variable	-	P	P	P	-	P
	* or /	P	-	P†	-	P	-
	- or +	P	-	-	-	P	-
	**	P	-	P†	-	P	-
	(	P	-	P	-	P	-
	)	-	P	P	P	-	P

†This is permitted when + or - indicates the sign of a numeric literal.

ARITHMETIC STATEMENTS

Basic arithmetic operations are specified by ADD, SUBTRACT, MULTIPLY, and DIVIDE. With COMPUTE, the user may specify any of the basic operations with arithmetic expressions.

Internal arithmetic operations are usually performed in the binary mode, except when both addition or subtraction operands are defined as display. If the composite size is ten or less, display operands are not converted and object time subroutines are provided to do the arithmetic in Display Code. For computational-1, single precision arithmetic operations are performed on operands of 14 digits or less, and double precision on operands of 15 to 18 digits. Single precision floating point operations are performed on computational-2 operands. (See USAGE for difference between computational-1 and computational-2.)

Rules for Arithmetic Verbs

The following general rules apply to all five arithmetic verbs:

All literals specified in arithmetic statements must be numeric. Wherever it is legal to specify a literal, the figurative constant ZERO(S) (ES) may be used. Other figurative constants, including ALL any-literal, may not be used since they are considered alphanumeric.

An identifier in an arithmetic statement must be a numeric elementary item.

The maximum size of an operand is 18 decimal digits. If the entry in the Data Division for an operand specifies a size greater than 18 digits, an error is indicated at compilation time.

No rounding will take place if the receiving field has more places to the right of the decimal point than is indicated by the pictures of the intermediate result.

The items in an arithmetic statement may be mixed in usage, (computational, computational-n, or display) as long as they are all numeric. Any necessary linkage to conversion routines is supplied by the compiler. Decimal point alignment is supplied automatically throughout computations.

No item used in computations may contain symbols; such an item will produce a compilation diagnostic. Operational signs and assumed decimal points are not editing symbols. An item that is to receive results may contain editing symbols if it is not used in subsequent computations as an operand. When an item that receives results contains editing symbols, the result is edited before it is moved to the item.

4.7 OPTIONS

Several options appear frequently in the statement descriptions that follow: ROUNDED, ON SIZE ERROR, and CORRESPONDING. The latter option is discussed specifically in the ADD, SUBTRACT, and MOVE statements. ROUNDED and ON SIZE ERROR options apply to all arithmetic statements.

ROUNDED Option

If the number of decimal positions (right of the decimal point) in a computed result exceeds the number of decimal positions in the format of the item receiving the result, truncation occurs when the object program is executed. Excess digits are dropped in accordance with the format of the item storing the result. If the ROUNDED option is specified following the name of a result item, excess digits are truncated and rounding takes place; the least significant digit specified by the format of the result item is increased by 1 when the most significant truncated digit is 5 or greater.

The degree of accuracy (the number of significant decimal positions) depends on the operation and the operands involved. For example, if a result item is specified with 5 decimal positions (point location is 5 left) the degree of accuracy is up to 5 positions beyond the decimal point. If an operation does not produce the accuracy requested, the remaining positions are filled with zeros. For example, if an operation produces a degree of accuracy of 2 and the result item is specified as above, the low-order 3 positions are filled.

ON SIZE ERROR Option

This option applies only to size errors in the final result. When the number of integral positions (left of the decimal point) in a COMPUTE result might exceed the number of integral positions in a receiving item, a diagnostic at compilation time specifies an apparent error in the program. Because the processor takes into account the maximum number of digits that could occur as the result, this diagnostic does not necessarily imply that there will be an error. Action at object program execution time depends on the ON SIZE ERROR specification.

1. When the result field is defined as decimal and the ON SIZE ERROR clause is specified, if a computed result appears to have more integral positions than are permitted in the result field, the high order excess digits are tested for being equal to zero. If they are all equal to zero, the computed result is delivered to the result field, zero digits are truncated, and no size error is considered to have occurred. If excess digits are not all equal to zero, the result field retains its current value and does not receive the computed result; a size error is considered to have occurred.

2. When the result field is binary floating point and the ON SIZE ERROR clause is specified, a size error is considered to have occurred only if the exponent is out of range or indefinite.

This option applies to size errors in the final result only.

4.8 PROCEDURE DIVISION STATEMENTS

All statements used in the Procedure Division (except Report Writer statements) are described in alphabetical order below. The Report Writer statements are described in Chapter 5.

This verb causes the transfer of low-volume data from an appropriate external device to the computer.

Format 1:

ACCEPT identifier-1

Format 2:

ACCEPT identifier-2 [FROM mnemonic-name]

Format 1 is used when data is to come from the system file, INPUT. If data is to be accepted from a file other than the system file, INPUT, format 2 must be specified. The mnemonic name must be equated to a SCOPE file name in the Environment Division (implementor-name IS mnemonic-name clause of the Special-Names paragraph). This mnemonic-name is effectively a file and is counted in determining the maximum number (53) of files and reports allowed for a COBOL program. The file used by ACCEPT (INPUT or mnemonic-name) must not be used in any other part of the COBOL program or a fatal error will result.

The identifier specified with the ACCEPT verb can be the name of a group or an elementary item and it can be subscripted. The input format must be that of an 80 column card in BCD form. If the identifier is 80 characters or less, they are the first characters on the card image. If the size is greater than 80 characters, several card images will be read and moved to the identifier until the number of characters specified in the description of the identifier has been accepted. If there is any remaining space on the last card, it is inaccessible; the next ACCEPT statement will begin at column 1 of a new card image.

This verb adds two or more numeric values and sets the result as the value of one or more items.

Format 1:

```
ADD {identifier-1} [{identifier-2}] ...identifier-n [ROUNDED] [ON SIZE ERROR
    {literal-1   }  [{literal-2   }]                                imperative-statement]
```

Format 2:

```
ADD {identifier-3} [{identifier-4}] ... {GIVING} identifier-m [ROUNDED] [identifier-n
    {literal-3   }  [{literal-4   }]    {TO    }
[ROUNDED]]...[ON SIZE ERROR imperative-statement]
```

Format 3:

```
ADD {identifier-5} [{identifier-6}] ... TO {identifier-7} GIVING identifier-m [ROUNDED]
    {literal-5   }  [{literal-6   }]       {literal-7   }
[identifier-n [ROUNDED]] ..[ON SIZE ERROR imperative-statement]
```

Format 4:

```
ADD CORRESPONDING identifier-8 TO identifier-9 [ROUNDED] [identifier-10 [ROUNDED]]...
[ON SIZE ERROR imperative-statement]
```

The composite size of the operands cannot exceed 18 digits. If this limit is exceeded, a diagnostic will be printed at compile time, and at object program execution time, the result may be unpredictable.

Rounding cannot take place if the number of places to the right of the decimal point in the receiving field is greater than or equal to the number of places to the right of the decimal point of the intermediate result.

An ADD statement must name at least two addends: the minimum ADD statement is ADD literal-1 identifier-2 or ADD [CORRESPONDING] identifier-8 TO identifier-9.

When the first format is specified, the literals and/or the values of the items named (including identifier-n) are summed and the result is set as the value of the identifier-n item.

The second format stores the result of an addition as the value of one or more items. When GIVING is specified, the literals and/or the values of the items preceding GIVING are summed and the result is set as the value of the identifier-m item. The same result is also set as the value of all other items after identifier-m (such as identifier-n). When TO is specified, the literals and/or the values of the items preceding TO are summed, and the total is added to the value of the identifier-m item; the final result is set as the value of the identifier-m item. The same total is added to the value of each item named after identifier-m (such as identifier-n) and the final result in each case is set as the value of the named item.

The third format stores the result of an addition as the value of one or more items. The literals or the values of the items named preceding TO and the identifier-7 or literal-7 following TO are added. The total is set as the value of the identifier-m item and all following items.

In formats 1, 2, and 3 all items named must be described in the Data Division as elementary numeric items. If a name encountered during compilation is not an elementary item, the processor issues a diagnostic and compilation of the ADD statement terminates. All literals specified must be numeric.

ADD CORRESPONDING

The fourth format adds the values of one or more elementary items within a group to the values of one or more selected elementary items within other groups. No literals may be included in this format, and all identifiers must be the names of groups. The groups are considered in pairs for the process of selecting one or more items from within each group to be added. The identifier-8 group is paired in turn with each of the other groups named. For any pair of groups, the values of a selected pair of elementary items are added and the result is set as the value of the selected item in the identifier-9 group and all following groups.

A pair of items from a pair of groups is selected for addition if:

a. Their names are the same

b. The names of all higher-level items in each group (the qualifiers for each item), up to but not including the names of the groups, are identical

c. Both items are elementary

After a pair has been selected for addition, the operation is the same as if an ADD statement of the second option using TO had been specified. The sames rules apply to the addition of the selected items as to addition of items specified by the user.

After all matched pairs have been added for the identifier-8 and identifier-9 groups, the matched pairs are added for the identifier-8, identifier-10 groups, and so on.

The following rules apply to ADD CORRESPONDING only:

1. No item described by an OCCURS clause in the Data Division can be involved in the addition.
2. Within a specified group, any item with a REDEFINES clause in the Data Division entry is ignored, as are all items within the same redefinition.
3. Independent data items with level number 77 cannot be referenced.
4. Within a specified group, any item with a RENAMES clause in the Data Division entry is ignored.

Examples:

```
ADD MONTHLY-EARNINGS OVERTIME-EARNINGS GROSS-YEAR-TO-DATE

ADD MONTHLY-EARNINGS OVERTIME-EARNINGS GIVING MONTHLY-GROSS-PAY
WORK-MONTHLY-GROSS-PAY

ADD HOS-INSURANCE LIFE-INSURANCE STATE-UNEMPLOYMENT UNITED
MISCELLANEOUS GIVING TOTAL-DEDUCTIONS

ADD CORRESPONDING UPDATE-RATE-TABLE TO RATE-TABLE ON SIZE ERROR
PERFORM RATE-OVERFLOW-PROC
```

UPDATE-RATE-TABLE and RATE-TABLE are described as follows:

```
01  UPDATE-RATE-TABLE
    03  EASTERN-REG
        05  NEW-YORK
            07  RATE
        05  BOSTON
            07  RATE
    03  WESTERN-REG
        05  LOS-ANGELES
            07  RATE
```

```
01  RATE-TABLE
    03  EASTERN-REG
        05  NEW-YORK
            07  RATE
        05  BOSTON
            07  RATE
        05  PHILADELPHIA
            07  RATE
                :
                :
    03  WESTERN-REG
        05  LOS-ANGELES
            07  RATE
        05  SAN-FRANCISCO
            07  RATE
                :
                :
    03  MIDWEST-REG
        ...
```

The rates for New York, Boston, and Los Angeles in the RATE-TABLE are added to the rates for these three cities in the UPDATE-RATE-TABLE, and the results are the new rates. No other alteration occurs in the RATE-TABLE.

If one or more of the additions results in a size error, the procedure RATE-OVERFLOW-PROC is performed after completion of the whole ADD statement. Each receiving item that has a size error retains its current value instead of the computed result. Control returns to the statement following ADD statement.

The purpose of this verb is to complete the meaning or modify the effect of one or more GO TO statements specified elsewhere in the program.

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

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

Procedure-name-1 is the name of a paragraph which contains the GO TO statement to be altered. Similarly, procedure-name-3 is the name of a paragraph which contains another GO TO statement to be altered.

The effect of the ALTER statement is to insert procedure-name-2 as the procedure name in the GO TO statement in the procedure-name-1 paragraph, if this statement does not contain an explicit procedure name. If it does contain an explicit procedure name, the effect of the ALTER statement is to replace that name by procedure-name-2. Since a GO TO statement is meaningless without a procedure name, the ALTER statement which supplies the name must be executed before the GO TO statement is executed. A maximum of 100 procedure-names may appear in an ALTER statement.

Examples:

```
        ALTER CC1 TO PROCEED TO CC5; CC5 TO PROCEED TO FINAL-RESULT.
CC0.    ADD 001 TO COUNTR.
        IF COUNTR IS LESS THAN OVFLW GO TO CC2.
CC1.    GO TO.
CC2.    MOVE CORRESPONDING INPUT-TABLE TO WORK-AREA.
        ADD INP1 OF WORK-AREA TO I-TOTAL.
        ADD INP2 OF WORK-AREA TO P-TOTAL.
        GO TO CC0.
CC5.    GO TO CC10.
          .
          .
          .
```

When the ALTER statement is executed, the paragraph name CC5 is inserted as the object of the GO TO in paragraph CC1; and the paragraph name FINAL-RESULT is inserted in place of CC10 as the object of the GO TO in paragraph CC5. FINAL-RESULT and CC10 must be procedure names in the COBOL program.

When processing has been completed, the file must be made unavailable for further use, or closed. Before a file can be closed, it must have been opened by an OPEN statement.

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

Each file named in a CLOSE statement must be defined in a File Description entry (FD) in the Data Division of the source program. The CLOSE statement can name any number of files; at least one must be named. If the file is specified as optional in the FILE-CONTROL, paragraph of the ENVIRONMENT division, the closing operations will not be performed if the file is not present at object program execution time.

When a CLOSE statement without any options is executed, the file is rewound. If the label record is standard, an end-of-file label is written before the file is rewound. If the NO REWIND option is included, none of the currently mounted reels is rewound. If the LOCK option is included, the file is rewound with interlock, making it unavailable for further use until operator action is taken.

The function of the statement differs if the file referenced by CLOSE is associated with one of the system files: INPUT, OUTPUT, PUNCH, and PUNCHB. The reel is not rewound and an ending label is not present; all other processing is performed. If the LOCK option is used, the reel will not be rewound, but all further references to the file will be ignored. That is, the system file will be closed but not rewound.

End-of-file procedures specified by the USE verb are executed before or after the checking or writing of the label record, depending on the specifications in the USE statement.

The REEL option of the CLOSE statement may be used to stop processing on a particular reel of a multiple-reel tape file (input or output) before the normal end. CLOSE... REEL has no meaning for the system units INPUT, OUTPUT, PUNCH, and PUNCHB. CLOSE... REEL does not affect the whole file; it applies only to the reel currently being processed. For an input file, further processing of the current reel is prevented. For an output file, the end-of-reel label is written if label records are standard and the file is not a system unit. Ending label procedures specified by the USE statement are executed before or after writing the label record, depending on the USE statement. Then CLOSE... REEL causes the next reel in sequence to be mounted for reference by subsequent statements. It checks or writes any standard beginning-of-reel label records for the new reel and executes any beginning label procedure specified by the USE statement. Procedures for checking or writing non-standard beginning or ending labels are discussed under USE.

Each reel terminated by the REEL option is rewound unless NO REWIND is specified. If LOCK is specified, the reel is rewound with interlock, making it unavailable for further use. CLOSE... REEL is meaningless for the last reel of a file since the final reel should be controlled by the CLOSE statement for the file. When the CLOSE statement is not specified for each reel, the input-output control system automatically performs the necessary actions.

Arithmetic operations on numeric values may be specified by arithmetic expressions. The final result of the operations is set as the value of one or more items.

COMPUTE identifier-1 [ROUNDED] [identifier-2 [ROUNDED]]... { FROM / = / EQUALS }

{ literal / any-arithmetic-expression / identifier-3 } [ON SIZE ERROR any-imperative-statement]

The arithmetic expression is first evaluated according to the rules in 4.6. The resulting numeric value is set as the value of the identifier-1, and all subsequent items. FROM, =, or EQUALS are chosen according to readability; they are equivalent in meaning.

All identifiers in the statement (including those in the arithmetic expression) must be described in the Data Division as elementary numeric items. If the compiler encounters a name that is not an elementary item, it issues a diagnostic and compilation of the COMPUTE statement terminates. Literals specified in the arithmetic expression must be numeric.

The arithmetic expression may be simple or complex. If it consists of a single identifier or numeric literal, the COMPUTE statement is equivalent to a MOVE statement, and the identifier-1 item is set to the value of this single item.

Most operations that can be specified by ADD, SUBTRACT, MULTIPLY, and DIVIDE statements can also be specified by the COMPUTE statement.

Example:

```
COMPUTE COST-PRICE FROM (HOURS * RATE + PARTS-COST) * (1 + PROFIT-FACTOR)
```

The intermediate results of a COMPUTE statement are discussed in Appendix D.

This verb causes data to be transferred in low volume from the computer to an output file or the system console.

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

When the DISPLAY statement is executed, the character in the identifiers or any literals specified are displayed consecutively. Spaces appear between characters only when spaces are specified as characters. The number of characters specified as identifiers and/or literals in any one DISPLAY statement may not exceed 72. Any characters over 40 are continued on the second line of the dayfile; the dayfile is the system history output with the normal listing. No more than five identifiers may be included in one DISPLAY statement; any number of literals, including figurative constants, may be specified within the limit of characters allowed.

If a mnemonic-name is used, it must be defined as a SCOPE file name or as the CONSOLE in the Special-names paragraph of the Environment Division. Either implementor-name IS mnemonic-name or CONSOLE IS mnemonic-name may be used. When mnemonic-name is not included in the DISPLAY statement, data is automatically sent to the OUTPUT file.

The identifier may name a group or an elementary item, and it may be subscripted. If an identifier is defined as computational-n, results are unpredictable. A literal in a DISPLAY statement may be numeric or non-numeric.

The first position following the quotes in a non-numeric literal is reserved for a carriage control character. This position should not contain part of the message as it will be lost.

The carriage control characters are:

(blank)	space 1 line
0	space 2 lines
1	page eject
+	no advance

Any other character has the same meaning as blank.

This verb divides one numeric value into one or more numeric values and sets the quotient as the values of one or more items.

Format 1:

DIVIDE {identifier-1 / literal-1} INTO identifier-2 [ROUNDED]

[identifier-3 [ROUNDED]] ... [ON SIZE ERROR any-imperative-statement]

Format 2:

DIVIDE {identifier-4 / literal-2} INTO {identifier-5 / literal-3} GIVING identifier-6 [ROUNDED] [identifier-7 [ROUNDED]] ... [ON SIZE ERROR any-imperative-statement]

Format 3:

DIVIDE {identifier-8 / literal-4} BY {identifier-9 / literal-5} GIVING identifier-10 [ROUNDED] [identifier-11 [ROUNDED]] ... [ON SIZE ERROR any-imperative-statement]

In the first format one or more divide operations may be specified by a single statement, with the quotients stored as the values of different items. Literal-1 or the value of identifier-1 is divided into the value of identifier-2; the quotient is set as the value of identifier-2. A similar quotient is formed with the value of identifier-3 as the dividend, and the result set as the value of this item, and so on for every item named in the statement.

The second format stores the result of a division as the value of one or more items. Literal-2 or the value of identifier-4 is divided into literal-3 or the value of identifier-5, as specified, the quotient is set as the value of identifier-6 and also as the value of all items named after identifier-6.

As in the second format, the third format stores the result of a division as the value of one or more items. Literal-4 or the value of identifier-8 is divided by literal-5 or the value of identifier-9.

All items must be described in the Data Division as elementary numeric items. If the compiler encounters a name that is not an elementary item, it issues a diagnostic and compilation of the DIVIDE statement terminates. Literals specified in the statement must be numeric, and they cannot be specified as dividends in the first option.

The identifier to the right of the word GIVING, in Formats 2 and 3, may be an edited numeric item.

The composite size of items for which a result is stored must not exceed 18 decimal digits. The ROUNDING option may be specified. Example:

```
DIVIDE 1.8 INTO CONVERTED-TEMP1 CONVERTED-TEMP-2
CONVERTED-TEMP3 CONVERTED-TEMP4
```

The ENTER statement allows the COBOL program to branch to a closed assembly language subroutine during execution of the object program.

$$\underline{\text{ENTER}}\ \text{subroutine-name-1}\ \left\{\begin{array}{l}\text{[data-name-1 [data-name-2]...]}\\ \text{[procedure-name-1 [procedure-name-2]...]}\\ \text{[file-name-1 [file-name-2]...]}\end{array}\right\}$$

The subroutine name is the entry point and must be defined by the subroutine. It may not exceed seven characters. The first six characters must be unique, with the first alphabetic. The data names are used for transmittal of data between the COBOL program and the subroutine. They must be defined in the Data Division and may be subscripted but the subscript must be a literal. The procedure names are entry points in the Procedure Division of the COBOL program to which the subroutine may return. They must conform to the SCOPE standards for entry point names; the first six characters must be unique with the first alphabetic. The file names are files, defined by File Description Entries in the COBOL program, which may be used for input or output by the subroutine entered.

The need for data names, procedure names, or file names depends entirely on the subroutine to be entered and the use made of it. Any subroutine that can be loaded can be referenced by the ENTER statement. Appendix E contains a description of the calling sequence generated by the ENTER statement.

ENTRY defines entry points for separately compiled COBOL subprogram.

ENTRY procedure-name-1 [procedure-name-2]...

The procedure names are used as entry points in separately compiled subprograms (Subcompile Capability, Chapter 6). The procedure names must be unique in the first six characters with the first alphabetic.

ENTRY is not an executable statement and must be the first entry in the Declaratives Section of the subprogram containing the procedure names.

Example:

COBOL program A contains the following statement:

```
PERFORM BB1.
```

If BB1 is in program B, an ENTRY statement is included in the Declaratives in B:

```
PROCEDURE DIVISION.
DECLARATIVES.
    ENTRY BB1.
END DECLARATIVES.
    :
    :
BB1....
```

The EXAMINE verb is used to replace a character in a data item and/or to count the number of times a character appears in the item. Each character in the item is examined beginning at the left.

```
                         ⎧           ⎧ALL        ⎫                                ⎫
                         ⎪TALLYING   ⎨LEADING    ⎬ literal-1                      ⎪
                         ⎪           ⎩UNTIL FIRST⎭ [REPLACING BY literal-2]       ⎪
EXAMINE identifier-1     ⎨           ⎧ALL           ⎫                             ⎬
                         ⎪REPLACING  ⎨LEADING       ⎬ literal-3 BY literal-4      ⎪
                         ⎪           ⎩[UNTIL] FIRST ⎭                             ⎪
                         ⎩                                                        ⎭
```

TALLYING ALL Occurrences of the literal-1 character in the item are counted and the result is stored as the current value of an 8-character special register, TALLY, which can be referenced by name.

TALLYING LEADING Occurrences of the literal-1 character prior to a character other than literal-1 are counted, and the count is stored in TALLY.

TALLYING UNTIL FIRST The number of characters other than literal-1 which occur before the literal-1 character are counted, and the count stored in TALLY.

The REPLACING option used in conjunction with the TALLYING option produces the following additional effects:

For TALLYING ALL, literal-2 is substituted for each occurrence of literal-1.

For TALLYING LEADING, literal-2 is substituted for each leading literal-1. Substitution terminates upon the occurrence of a character other than literal-1.

For TALLYING UNTIL FIRST, literal-2 is substituted for every character in the item until the first occurrence of a literal-1 character.

REPLACING ALL A literal-4 character is substituted for each occurrence of literal-3 in the item.

REPLACING LEADING A literal-4 character is substituted for each leading literal-3 character, until the occurrence of a character other than literal-3.

REPLACING UNTIL FIRST A literal-4 character is substituted for every character in the item until the first occurrence of a literal-3 character.

REPLACING FIRST A literal-4 character is substituted for only the first appearance of literal-3.

Each literal may consist of only 1 character which is a member of the set associated with the CLASS of the named item. If CLASS is NUMERIC the literal may be only 0-9, for ALPHABETIC it may be only A-Z or blank, for ALPHANUMERIC it may be any character in the COBOL character set. A non-numeric literal must be enclosed in quotation marks. Every group item is treated as an alphanumeric item. No special rules apply to non-numeric elementary items. Plus and minus signs are stripped from numeric elementary items before examination.

This verb provides a common end point for a procedure executed as a result of a PERFORM statement. The statement format is simply:

EXIT.

This statement must be terminated with a period; it stands by itself as a paragraph and is assigned a procedure name. EXIT is not needed when there is only one possible exit from a procedure performed as a result of a PERFORM statement. When one or more conditional exits are possible, EXIT must be specified following the last significant paragraph of the procedure, to ensure that all paths through the procedure have the same ending point. The PERFORM statement must name the EXIT paragraph, so that EXIT is the last statement executed. Any conditional statement that involves an exit from the procedure must also name the EXIT paragraph as the place to which control transfers when the exit branch of the conditional is taken. Irrespective of the results of conditional statements, all exits from the procedure are the same; namely, following the execution of the EXIT paragraph. Since any READ statement, or a WRITE with the INVALID KEY clause is a conditional statement, the user must insure that the imperative statement following either clause contains a proper exit whenever such a statement is used within a procedure controlled by PERFORM.

In reality, since EXIT is a processor-directing verb, the processor does not compile instructions from a statement using it. The purpose is to return control to the statement following the PERFORM statement. The processor sets up the control instructions whenever the EXIT paragraph is referenced, directly or in-line, under control of a PERFORM statement. If such control instructions are encountered when the procedure is executed normally and not under control of a PERFORM statement, they have no effect and do not cause transfer of control. If EXIT is not alone in a paragraph, the complete paragraph will be compiled and executed.

This verb specifies a permanent transfer of control to another point in the program.

Format 1:

GO TO [procedure-name-1].

Format 2:

GO TO procedure-name-1 [procedure-name-2]...DEPENDING ON identifier-1.

The first format results in an unconditional transfer of control to the beginning of the paragraph or section specified by the procedure name. This name may be omitted when it is supplied by an ALTER statement prior to execution of the GO TO statement. If, at execution time, this GO TO statement is not completed by an ALTER statement before execution, an error diagnostic is produced and the job is terminated.

The ALTER statement can be used to supply the procedure name in a GO TO statement or to alter a specified procedure name in a GO TO statement. A statement to be altered in either way should comprise a paragraph by itself. The paragraph is assigned a procedure name so that the ALTER statement can reference to GO TO statement. A normal GO TO statement (when the procedure name is not to be modified) can appear as one of several statements in a paragraph, but it can be used only as the final statement in a sequence because of its sequence changing effect. If it appears in any other position, statements following it would be bypassed.

The second format permits transfer of control to one of several procedures in the program, depending on the value of a particular data item at execution time. Control is transferred to the paragraph or section specified by procedure-name-1, -2, or n, depending on whether the item value is equal to 1, 2, or n. The identifier-1 value must be a positive integer. If the item value is not within the range 1 through n, no transfer occurs: instead, control passes to the statement following the GO TO. A maximum of 100 procedure names may be used in one GO TO statement. The identifier may be of any USAGE, and it may be subscripted.

Example:

```
ADD 1 TO COUNT.
IF COUNT GR 3 GO TO CONTINUE.
GO TO PR1 PR2 PR3 DEPENDING ON COUNT.
```

The IF statement defines operations to be executed or bypassed depending on whether a stated condition is true or false. A condition is evaluated, and subsequent sequence of execution depends on whether the condition is true or false.

$$\underline{\text{IF}}\ \text{conditional-expression}\ [\text{THEN}]\ \left\{\begin{matrix}\text{statement-1}\\ \underline{\text{NEXT SENTENCE}}\end{matrix}\right\}\ \left[[\text{THEN}]\ \left\{\begin{matrix}\underline{\text{OTHERWISE}}\\ \underline{\text{ELSE}}\end{matrix}\right\}\ \left\{\begin{matrix}\text{statement-2}\\ \underline{\text{NEXT SENTENCE}}\end{matrix}\right\}\right] .$$

An IF statement may be preceded by an imperative statement, which is executed regardless of the result of the conditional statement. Both statement-1 and -2 can be one or more imperative statements or a conditional statement, or both. Either statement-1 or -2 may contain conditional statements that contain conditional statements, and so on, to the depth of 25 levels. Conditional statements contained in other conditional statements are said to be nested.

The phrase NEXT SENTENCE can be specified in place of either statement-1 or -2 or both. The conditional expression is evaluated first. If true, statement-1 or NEXT SENTENCE is executed and control passes to the beginning of the next sentence. If the conditional expression is false, statement-2 or NEXT SENTENCE is executed followed by the execution of the remainder of the sentence.

A sentence may contain any number of independent conditional statements, nested or not. If the phrase OTHERWISE (or ELSE) NEXT SENTENCE directly precedes the period that terminates the sentence, the whole phrase may be omitted. This rule may then be applied to the resulting sentence, and so on. In this manner, the simplest form of the conditional sentence (IF conditional-expression imperative-statement) may be derived from the general form.

A discussion of the interpretation and specification of conditional expressions is given in section 4.5.

Examples:

1) IF COUNTER IS GREATER THAN 5 GO TO RESET ELSE ADD 1
   TO COUNTER GO TO UPDATE-PROC.

2) IF A IS NUMERIC MULTIPLY A BY B THEN IF B IS LESS THAN 50
   ADD A B TO C ELSE ADD A B TO D ELSE GO TO BADCLASS.

3) IF PERFORM-COUNT IS POSITIVE GO TO START ELSE NEXT SENTENCE.

4)
```
DATA DIVISION.
    01  PASSING-GRADE.
        88  PASS VALUE 75 TO 100.
        88  FAIL VALUE 0 TO 75.
                :
                :
PROCEDURE DIVISION.
                :
                :
IF PASS GO TO X.
                :
                :
```

The INCLUDE statement is used to incorporate routines from the COBOL source library into the Procedure Division of the source program.

```
INCLUDE procedure-name [ REPLACING { literal-1       } BY { literal-2       }
                                   { routine-word-1  }    { routine-word-2  }
                                   { identifier-1    }    { identifier-2    }
                       [ { literal-3       } BY { literal-4       } ] ... ].
                       { routine-word-3  }    { routine-word-4  }
                       { identifier-3    }    { identifier-4    }
```

The INCLUDE statement must always be in a sentence by itself. The procedure-name is a library routine, that is, a procedure stored in the COBOL source library. This library routine may either be one paragraph or one section which is incorporated into the source program at compile time for in-line execution at object time. If the library routine is a single paragraph, INCLUDE must be the only statement in the source program paragraph; if the library routine is a single section, INCLUDE must be the only statement in the source program section.

When a paragraph is included it replaces the INCLUDE statement; but the paragraph-name associated with INCLUDE replaces the name of the library routine. The name of the section containing the INCLUDE statement is the only qualifier for the procedure names within the library routine. If the routine is a section, it replaces the INCLUDE section in the source program; but the section name and priority of the INCLUDE statement replace those of the library routine. When an included paragraph is subordinate to a section, the paragraph-name must be qualified by the section-name even if the paragraph-name is unique to the library.

When INCLUDE is part of the DECLARATIVES portion of the Procedure Division:

- The library routine must be a USE declarative
- INCLUDE must be on the same line as and immediately following the section header.

The REPLACING option is used to specify which words, identifiers, and literals in the library routine are to be replaced by corresponding words, identifiers, and literals when the INCLUDE statement is executed.

A routine word is any word in the library routine that is neither qualified nor subscripted. It may be any COBOL word or user-defined word of 30 alphanumeric characters or less. An identifier may be qualified and/or subscripted. The replacement of an identifier includes all associated qualifiers and subscripts. The REPLACING option must result in correct COBOL syntax.

Example:

Source Program:

```
A SECTION.
123. INCLUDE X REPLACING C OF D BY H OF J OF K, Y BY MOVE, 3BY G, 5 BY ADD,4 BY F.
                :
```

Library routine.

```
X. ADD 1 TO B OF C OF D.  Y 1 TO 3 5 4 TO F.
MOVE 6 TO C OF D. IF F LESS THAN 100 GO TO X.
```

Resulting source routine:

```
ADD 1 TO B OF H OF J OF K. MOVE 1 TO G ADD F TO F.  MOVE 6 TO H OF J OF K.
IF F LESS THAN 100 GO TO 123 OF A.
```

The MOVE verb transfers data from one storage area or item to another. If the receiving item specifies editing symbols (Chapter 3), PICTURE and clause editing occurs concurrently with data movement.

$$\underline{\text{MOVE}} \left\{ \begin{matrix} [\underline{\text{CORRESPONDING}}] \text{ identifier-1} \\ \text{literal-1} \end{matrix} \right\} \underline{\text{TO}} \text{ identifier-2 } [\text{identifier-3}] \ldots$$

This statement moves the data in identifier-1 or the specified literal to identifier-2. Literal-1 may be a numeric, alphanumeric or figurative constant. Figurative constants, with the exception of ZERO(S)ES), are treated as alphanumeric items. The same information may be moved simultaneously to additional areas, as specified by identifier-3, and so forth. Such movement does not destroy original data, but copies it in the designated areas. Identifier-1 or literal-1 is the source item; identifier-2, -3, and so on, are the receiving items or areas. Both source and receiving items can be elementary items or groups. (For the purpose of the MOVE statement, a literal is considered an elementary item.) Four types of moves may be specified.

1. elementary item ⟶ elementary item
2. elementary item ⟶ group
3. group ⟶ elementary item
4. group ⟶ group

The manner in which the move is performed depends not only on the type of source and receiving items, but also on their classes:

Type of MOVE	Manner of MOVE
Elementary	When source and receiving items are both elementary and: a. source and receiving items are numeric, the move is performed according to rules for numeric items. b. source item is numeric and receiving item is numerically edited (a report item), the move is performed according to rules for edited items. c. source item is numeric, alphabetic, or alphanumeric, and receiving item is alphanumeric, the move is performed according to rules for alphanumeric items. d. the source item is alphabetic and the receiving item is alphabetic, the move is performed according to the rules for alphanumeric items.
Group	When either the source or the receiving item is a group, the move is performed according to rules for alphanumeric items. Data is moved without regard for the level structure, point location, and so forth; data is treated simply as a sequence of alphanumeric characters or binary bits.

Numeric Items

When the source data is moved to the receiving area, it is aligned according to its decimal point and the decimal point in the receiving area. If the source is an identifier, the decimal point can only be an assumed one. If the source is a literal, it is treated as if it were an identifier with an assumed decimal point. If no decimal point is specified for either the source data or the receiving item, the data is right justified. This is equivalent to decimal point alignment, assuming the decimal points to be after the rightmost position in the source data and the receiving item. If the source data contains a decimal point, but the receiving item does not, data is aligned by decimal point, with the decimal point in the receiving area assumed to be after the rightmost position.

Alignment by decimal points may result in the loss of leading or trailing digits, or both. If a MOVE statement will cause leading digits to be lost, the processor issues a diagnostic at compilation time. At object program time, however, the statement is executed normally, insofar as is possible, with the resulting loss of significance.

In all cases, whether or not the MOVE is legal, any positions in the receiving area not filled with data are automatically filled with zeros. Such a situation could arise because of the decimal point alignment, the difference in sizes between the source and the receiving items, or both.

Examples:

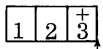
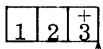
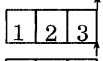
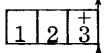
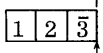
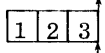
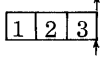
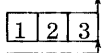
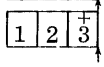
Source Data	Picture of Receiving Item	Receiving Item
1 2 3	99V9	1 2 3
1 2 3	999V99	0 1 2 3 0
1 2 3	9999	0 1 2 3
1 2 3	9999	0 0 1 2
-1.23 (literal)	S9V99	1 2 3̄
1 2 3	9V9	1 2
1 2 3	9V9	2 3

If source data is signed and the receiving item format specifies a sign, the operational sign is moved with data into the receiving area where it is stored as part of the low-order character. If the low-order character in the source is truncated because of decimal point alignment or size difference, the sign is stored as part of the low-order character actually stored in the receiving item. If source data is signed and the receiving item format does not specify a sign, it is stripped from the source data and not moved into the receiving area. If source data is not signed, but the receiving item format specifies a sign, source data is assumed to be positive, and a plus sign is stored as part of the low-order character in the receiving item.

Computational-1 and computational-2 items are always signed. They are stored in one or two computer words depending on whether its size is 14 digits or less; or 15 to 18 digits.

If a binary item is moved to a BCD field, or vice versa, conversion will be done by the MOVE statement.

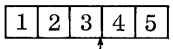
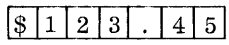
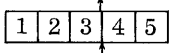
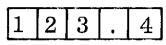
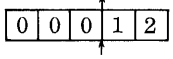
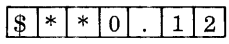
Examples:

Source Data	Picture of Receiving Item	Receiving Item
	S999	
	S999	
	999	
	999	
+123 (literal)	S999	

Edited items

If receiving item format specifies editing, source data is edited concurrently with data movement after decimal point alignment. Editing symbols in the receiving item (dollar signs, commas, and so on) make this item alphanumeric. If it is subsequently referenced as a source item in a MOVE statement, it is moved in accordance with the rules for alphanumeric items.

Examples:

Source Data	Picture of Receiving Item	Receiving Item
	$**9.99	
	999.9	
	$**9.99	

Further examples of editing are given in the discussion of the PICTURE clause, Chapter 3.

If the receiving item is numeric or numerically edited, the literal can be any numeric literal or the figurative constant ZERO(S)(ES).

Examples:

Source Data	Picture of Receiving Item	Receiving Item
+1.23	S9V99	1 2 +3
+1.23	S9V9	1 +2
123	9(5)	0 0 1 2 3
ZEROS	S99999	0 0 0 0 +0
QUOTES	9999	illegal (alphanumeric to numeric)
+37	S999V99	0 3 7 0 +0
ALL 37	S999V99	illegal (no quotes)
ALL "37"	S999V99	illegal (alphanumeric to numeric)
03737.3	$***9.9	$ 3 7 3 7 . 3

Alphanumeric Items

Source data is stored left justified in the receiving area unless the receiving item is an elementary item that specifies JUSTIFIED RIGHT. If a group is moved, left justification is standard; any specification to the contrary is overridden. If the receiving area is not completely filled by data, remaining positions are filled with spaces. If the receiving item is alphabetic, it is treated as alphanumeric.

Examples:

Source Data	Picture of Receiving Item	Receiving Item
A B C D	A(4) or X(4)	A B C D
A B C D	A(5) or X(5)	A B C D Δ
A B C 1 2 3	X(8)	A B C 1 2 3 Δ Δ
1 2 3	X(8)	1 2 3 Δ Δ Δ Δ Δ

If in a MOVE operation the receiving area is too small to contain the source data, the processor issues a diagnostic at compilation time. At object program execution, such a move is performed insofar as is possible, and terminates when the receiving area is filled.

Examples:

Source Data	Picture of Receiving Item	Receiving Item
A B C D E	A(4) or X(4)	A B C D
A B 1 2 3	X(3)	A B 1

If the receiving item is alphanumeric, the literal may be any literal or figurative constant. If the figurative constant takes the form of ALL any-literal, the literal must be enclosed in quotation marks; it is considered to be an alphanumeric item. Size of an ALL any-literal item is determined by size of the receiving item; the characters are repeated from left to right.

Examples:

Source Data	Picture of Receiving Item	Receiving Item
"ABCD"	X(4)	A B C D
ALL "ABCD"	X(7)	A B C D A B C
"ABC-123"	X(7)	A B C - 1 2 3
"123"	X(2)	1 2
ALL "123"	X(7)	1 2 3 1 2 3 1
ALL 123	X(7)	illegal (no quotes)
123	X(5)	1 2 3 Δ Δ
[ALL] †QUOTES	X(6)	" " " " " "
[ALL] †LOW-VALUES	X(4)	Δ Δ Δ Δ
[ALL] †HIGH-VALUES	X(5)	9 9 9 9 9
RECORD-MARK	X	]

It is illegal for an alphabetic or alphanumeric item to be moved to a binary numeric item or an item that contains numeric editing symbols. It is also illegal for a binary numeric item, or literal to be moved to an alphabetic item. If such moves are specified, the processor issues a diagnostic, and the move is performed or not according to the move matrix on the following page. A trivial diagnostic will be issued for an alphanumeric to alphabetic move.

† The word ALL may be used but is redundant; if ALL is omitted, the result is the same.

MOVE CORRESPONDING

The CORRESPONDING option in the MOVE statement is used to move one or more items within one group to selected areas within one or more other groups. The source area must be a data item (specified by identifier-1). The items moved depend on the names of the items within the source and the receiving items. An item is selected for movement if:

There is a like-named item in the receiving area.

The names of all higher-level items in each area (qualifiers for the pair of items) up to but not including identifier-1 and identifier-2, identifier-3, and so on, are also identical.

Each selected item is moved from the source area to the corresponding item in the receiving area; editing according to the receiving area format takes place concurrently with the move. The above rules for the simple MOVE statement apply to each move when the CORRESPONDING option is used. A MOVE CORRESPONDING statement has the same effect as a series of simple MOVE statements.

The following rules apply to the MOVE CORRESPONDING statement only:

1. No item described by an OCCURS clause in the Data Division can be involved in the move.
2. Data items with level number 77 (independent working-storage or constant items) cannot be referenced.
3. Within any group specified, an item with a REDEFINES clause in the Data Division entry is ignored, as are all items within the same redefinition.
4. Within a specified group, any item with a RENAMES clause in the Data Division entry is ignored.

Example:

If a record named MASTER contains information to be written as part of a record called MAST-REP, the MOVE CORRESPONDING statement can specify movement of all relevant data.

```
01  MASTER
    03  ITEM-NUMBER
    03  ITEM-NAME
        05  GENERAL-CLASS
        05  DETAIL-NAME
    03  ON-HAND-QUAN
    03  REORDER-LEVEL
    03  RETAIL-PRICE
    03  WHOLESALE-LEVEL
    03  WHOLESALE-PRICE
    03  YEAR-TO-DATE-SALES
    03  YEAR-TO-DATE-PROFIT
    03  LAST-YEAR-SALES
    03  LAST-YEAR-PROFIT

01  MAST-REP
    03  ITEM-NUMBER
    03  ITEM-NAME
        05  GENERAL-CLASS
        05  DETAIL-NAME
    03  YEAR-TO-DATE-SALES
    03  LAST-YEAR-SALES
    03  YEAR-TO-DATE-PROFIT
    03  PROJECTED-PROFIT
    03  LAST-YEAR-PROFIT
    03  SALES-COMP
    03  PROFIT-COMP
```

MOVE CORRESPONDING MASTER TO MAST-REP results in movement of items: ITEM-NUMBER, ITEM-NAME (GENERAL-CLASS and DETAIL-NAME), YEAR-TO-DATE-SALES, LAST-YEAR-SALES, YEAR-TO-DATE-PROFIT, and LAST-YEAR-PROFIT.

When a group containing synchronized items is referenced by a MOVE statement, the entire group is moved, including characters not used because of synchronization. If a synchronized elementary item is referenced, only the information is moved; unused positions are ignored. These rules also apply to each individual move resulting from a MOVE CORRESPONDING statement.

Source Field \ Type of Move \ Rec. Field	Elem. Binary Single-Prec.	Elem. Binary Double-Prec.	Elem. Alpha	Elem. BCD Num.	Elem. AN	Elem. Edit Num.	Group AN	Group Binary	Group Mixed	Fig. Cons. and Literal
Elem. Binary Single-Prec.	Num. Bin.	Num. Bin.	illegal	Conv. Num.	Conv.† AN	Conv. Edit	PD AN	PD AN	PD AN	illegal
Elem. Binary Double-Prec.	Num. Bin.	Num. Bin.	illegal	Conv. Num.	Conv.† AN	Conv. Edit	PD AN	PD AN	PD AN	illegal
Elem. Alpha	illegal	illegal	AN	PD AN	AN	illegal	AN	PD AN	PD AN	illegal
Elem. BCD Num.	Conv. Bin.	Conv. Bin.	PD AN	Num.	AN†	Edit	AN†	PD AN	PD AN	illegal
Elem. AN	illegal	illegal	PD AN	PD AN	AN	illegal	AN	PD AN	PD AN	illegal
Elem. Edit Num.	illegal	illegal	PD AN	illegal	AN	illegal	AN	PD AN	PD AN	illegal
Group AN	PD AN	PD AN	PD AN	PD AN	AN	illegal	AN	PD AN	PD AN	illegal
Group Binary	PD AN	PD AN	PD AN	PD AN	PD AN	illegal	PD AN	AN	PD AN	illegal
Group Mixed	PD AN	PD AN	PD AN	PD AN	PD AN	illegal	PD AN	PD AN	PD AN	illegal
Zero	Num. Bin.	Num. Bin.	illegal	Num.	AN	Edit	AN	AN	AN	illegal
Other Fig. Cons.	illegal	illegal	PD AN	illegal	AN	illegal	AN	PD AN	PD AN	illegal
Literal AN	illegal	illegal	PD AN	illegal	AN	illegal	AN	PD AN	PD AN	illegal
Literal Num.	Conv. Bin.	Conv. Bin.	illegal	Num.	AN†	Edit	AN	PD AN	PD AN	illegal

† Valid MOVE only when source is integer; others, precautionary diagnostic.
PD Precautionary diagnostic will be issued.

This verb multiplies two numeric values and sets the resulting product as the value of one or more items.

Format 1:

```
MULTIPLY {identifier-1} BY identifier-2 [ROUNDED] [identifier-3 [ROUNDED]]...
         {literal-1   }
                    [ON SIZE ERROR any-imperative-statement]
```

Format 2:

```
MULTIPLY {identifier-4} BY {identifier-5} GIVING identifier-6 [ROUNDED] [identifier-7
         {literal-2   }    {literal-3   }
                    [ROUNDED]]... [ON SIZE ERROR any-imperative-statement ]
```

In the first format one or more different multiplications may be specified by a single statement, with the products stored as the values of different items. Literal-1 or the value of identifier-1 is multiplied by the value of identifier-2; the product is set as the value of identifier-2. A product is formed with the value of identifier-3, and the result set as the value of this item, and so on for every item named in the statement.

The second format stores the product of a multiplication as the value of one or more items. Literal-2 or the value of identifier-4 is multiplied by literal-3 or the value of identifier-5, whichever is specified. The product is set as the value of identifier-6, and also as the value of all items named after identifier-6.

All items must be described in the Data Division as elementary numeric items. If a name encountered during compilation is not an elementary item, the processor issues a diagnostic and compilation of the MULTIPLICATION statement terminates. Literals specified must be numeric. Literals cannot be specified as multipliers in the first option.

The composite size of the items in which a result is stored must not exceed 18 decimal digits. The ROUNDING option may be specified.

Example:

```
MULTIPLY 1.05 BY MONTHLY-EARNINGS, OVERTIME-RATE,
    SOC-SEC, FEDERAL-TAX
```

This verb allows the user to write explanatory statements in the Procedure Division of the source program. They are reproduced in the source program listing but have no effect on the object program.

NOTE any-comment.

The word NOTE is followed by any combination of characters in the COBOL set; they may constitute a sentence or a paragraph, according to the following rules:

1. If NOTE is the first word of a paragraph, the entire text is considered notes. The paragraph must be named and proper format rules for the structure of the paragraph must be observed, however;
2. If NOTE is not the first word of a paragraph, the commentary is considered to end with the first period following the word NOTE. This period must be followed by a space before the first character of the next specification is written.

Example:

```
CC10.   PERFORM SUMMARIZE.
        NOTE  THIS PROCEDURE WILL SUMMARIZE THE FINAL RESULTS.

NOTE-PARAG.  NOTE SINCE THE FIRST WORD IN THE PARAGRAPH IS A NOTE, NO
             MATTER WHAT THE FOLLOWING SENTENCES SAY, THEY ARE ACCEPTED AS
             COMMENTARY ONLY.
```

Before data can be obtained from an input file or written on an output file by the READ and WRITE statements, the file must be made available by the OPEN verb.

```
        [ INPUT file-name-1 [ { REVERSED          } ]  file-name-2 [ { REVERSED          } ]  ... ]
OPEN    [ OUTPUT file-name-3 [WITH NO REWIND] [file-name-4 [WITH NO REWIND]]... ]
        [ { INPUT-OUTPUT } file-name-5 [file-name-6]... ]
        [ { I-O         }                               ]
```

Only one of each of the clauses (INPUT, OUTPUT, I-O or INPUT-OUTPUT) may be included in one statement. If two files are to share record and input-output areas, they should not both be open at the same time; the contents of these areas would then be dependent on the order in which READ or WRITE statements were issued for these files. An OPEN statement must be executed for a file before a READ or WRITE statement pertaining to that file can be executed.

The I-O and INPUT-OUTPUT options pertain only to mass storage files. These options permit the opening of a mass storage file for both input and output operations. It is illegal to OPEN a mass storage file specified as ACCESS MODE IS RANDOM if that file is assigned to a tape by a REQUEST card.

Each file named in an OPEN statement must be defined by a file (FD) description entry in the Data Division or by a RENAMING clause in the Environment Division. If label records are STANDARD, executing an OPEN statement causes the beginning-of-file label record to be checked for an input file or written for an output file. When label records are STANDARD, beginning procedures specified by the USE verb are executed before or after the checking or writing of the label record as specified. Refer to Chapter 6 for a further discussion on file labels.

A second OPEN statement mentioning the same file name cannot be executed unless it has been preceded by a CLOSE statement for that file.

The REVERSED option is not implemented at this time and will produce a diagnostic stating so.

NO REWIND may be specified when a file on a multiple file magnetic tape reel is opened. The reel is positioned at the beginning of the file. This option should be included for any file except the first on a multiple-file reel.

Examples:

```
OPEN INPUT CARD-FILE.
OPEN OUTPUT PRINT-FILEA WITH NO REWIND PRINT-FILEB.
OPEN INPUT-OUTPUT MASTER-FILE.
```

This verb specifies a temporary departure from the normal execution sequence to execute one or more procedures a number of times or until a specified condition is satisfied. A return of control is automatically set up.

When instructions compiled from a PERFORM statement are executed, they transfer control to the first statement of a specified procedure. Instructions set up during compilation return control to the statement following PERFORM after execution of the last statement in the procedure (paragraph, section, or combination of these).

The sentence executed just before control is returned should not contain a GO TO statement, otherwise any statements following it will be bypassed and return to the statement following PERFORM will never occur.

Execution sequence proceeds from the beginning of the procedure-name-1 to the last statement of the procedure-name-2. GO TO and PERFORM statements are permitted between these points, provided that the execution sequence ultimately returns to the final statement of the procedure-name-2.

If the PERFORM statement transfers control to a procedure which contains conditional exits before the end of the procedure, procedure-name-2 must be the name of an EXIT paragraph and every conditional exit must refer to this name. In this way, all of the foregoing requirements are satisfied and all paths through the procedure eventually lead to this EXIT paragraph. This paragraph must consist solely of a sentence containing only the verb EXIT. This ensures control is always returned to the statement following the PERFORM statement from every exit.

A procedure may be referenced by more than one PERFORM statement in the same program and it also may be executed in the normal sequence.

Format 1; simple PERFORM:

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

A procedure referenced by this statement is executed once, and control returns to the statement following the PERFORM statement.

Format 2; TIMES option:

PERFORM procedure-name-3 [THRU procedure-name-4] { identifier-1 / integer-1 } TIMES

In this option, a procedure is executed repetitively. The number of executions may be specified explicitly as an integer or it may be specified as the value of an elementary data item.

An identifier may be of any usage, and it may be subscripted, but it should be a positive integer.

When this option is included in a PERFORM statement, a counter is set up with a value equal to the value of the identifier-1 item or integer-1. Before each execution, the counter is tested against the value zero. If it is not equal to zero, the procedure is executed and the value of the counter

decreased by 1; the counter is again tested. When the value of the counter is zero, the procedure has been executed the specified number of times and control transfers to the statement following the PERFORM statement. Since the counter is tested before the first execution, the procedure is not performed for zero execution.

Format 3; UNTIL option:

PERFORM procedure-name-5[THRU procedure-name-6] UNTIL condition-1

In this option the number of times the procedure is executed is dependent on the truth or falsity of a condition rather than a stated value.

Condition-1 can be any simple or compound conditional expression, which is evaluated before the specified procedure is executed. If it is found to be false, the procedure is executed and the expression is evaluated again (since the values are altered by execution) and tested, this process is repeated until the conditional expression is found to be true, at which point control transfers to the statement following the PERFORM statement. If the conditional expression is found to be true when the PERFORM statement is encountered, the procedure is not executed.

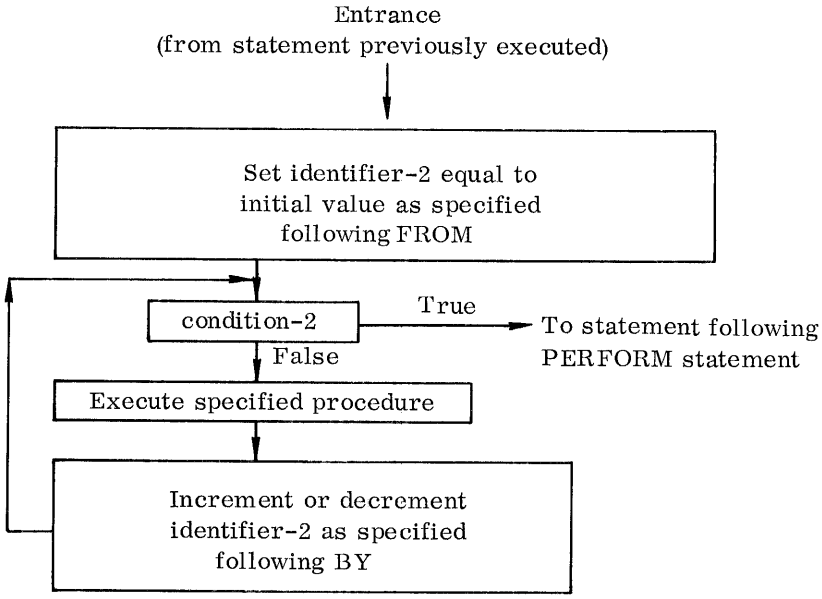
Format 4; VARYING option:

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

The VARYING option in the PERFORM statement is used to augment the value of one or more identifiers in a nested fashion in conjunction with repetitive executions of a particular procedure. A maximum of three levels may be varied in a given PERFORM statement.

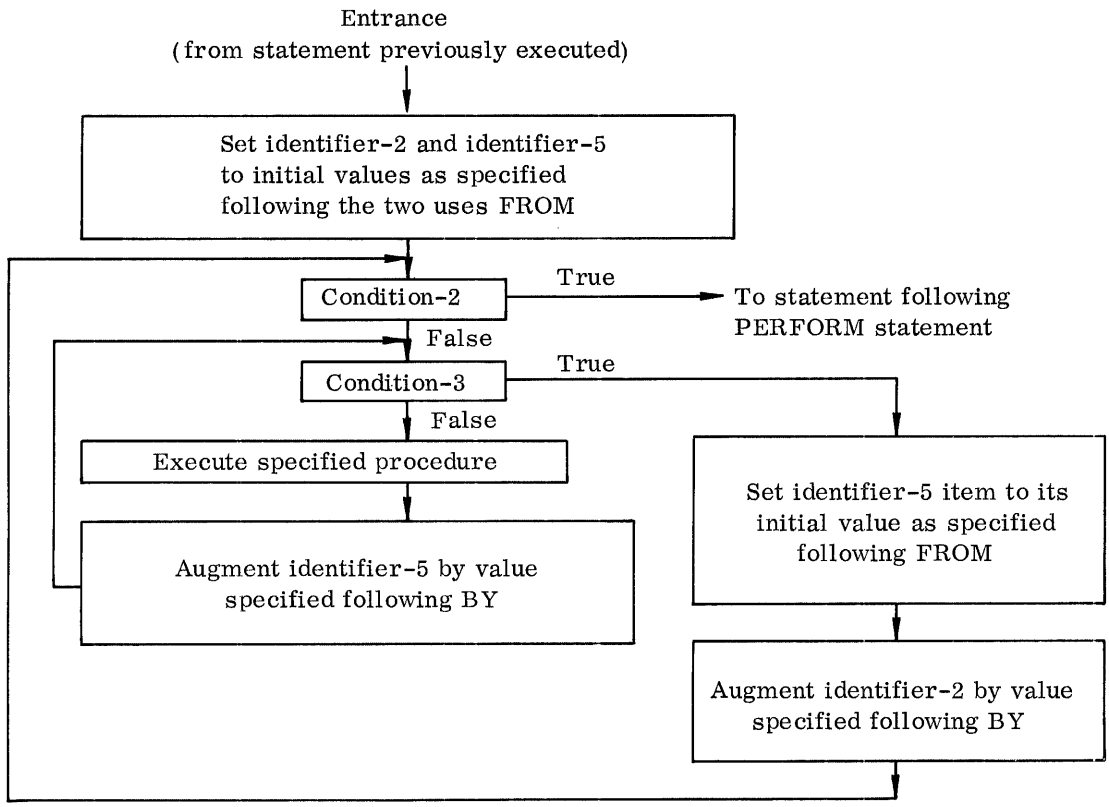
When one identifier is varied, identifier-2 is set equal to its starting value, identifier-3 or literal-1; if the condition is false, the sequence of procedures, procedure-name-7 through procedure-name-8, is executed once. The value of identifier-2 is augmented by the specified increment or decrement, identifier-4 or literal-2 and condition-2 is evaluated again. The cycle continues until this expression is true; at which point control passes to the statement following the PERFORM statement. If the condition is true at beginning of execution of the PERFORM, control passes directly to the statement following the PERFORM statement.

The following diagram illustrates the PERFORM statement with one varying identifier.



When two identifiers are varied (optional clause beginning with AFTER is included), the item associated with identifier-5 assumes every value in the range defined by FROM, BY, and UNTIL specifications, for every value that the identifier-2 item assumes in the range defined by the FROM, BY, and UNTIL specifications. When three identifiers vary, the identifier-8 assumes every value in the range for every value of identifier-5 and so on. For each combination of values, the specified procedure is executed once.

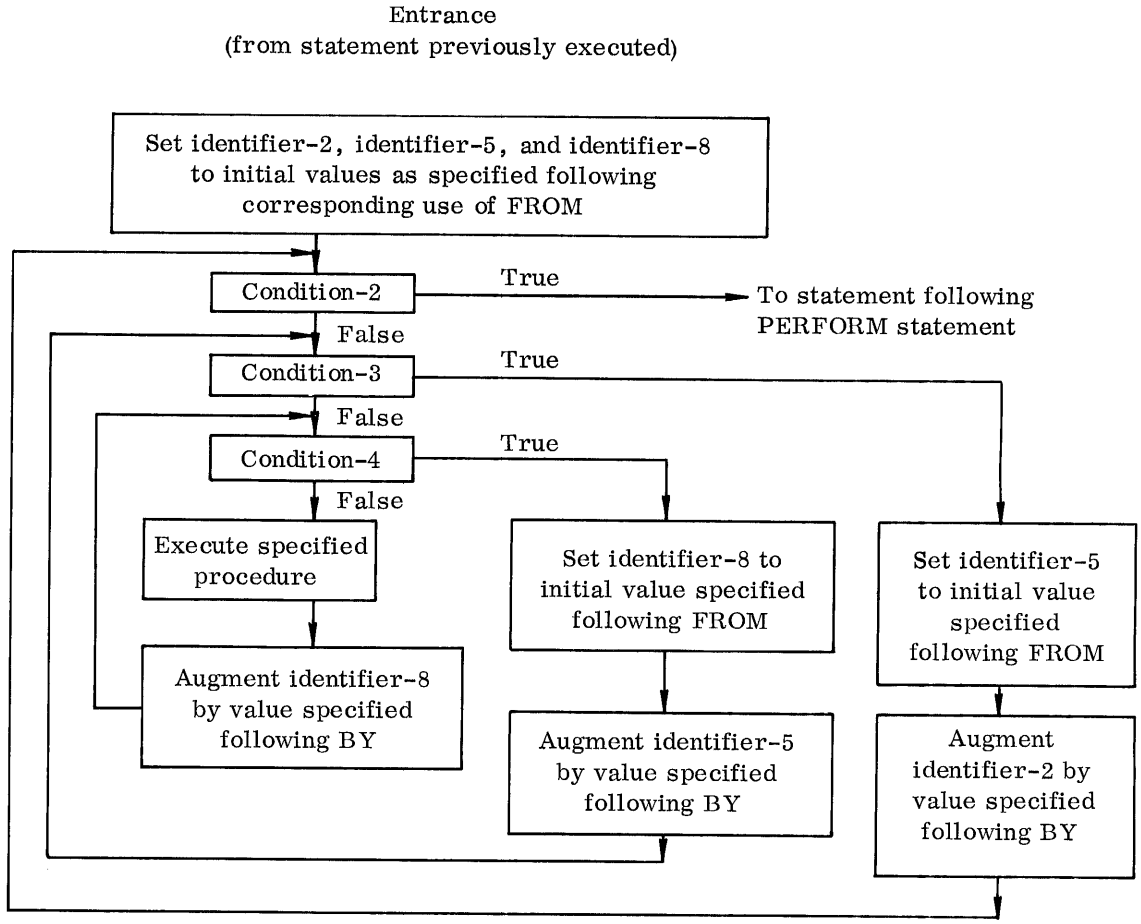
This diagram illustrates the PERFORM statement with two varying identifiers.



After the two identifiers are set to initial values, condition-2 is evaluated. If false, condition-3 is evaluated. If condition-3 is also false, the specified procedure is executed, identifier-5 is augmented in value and condition-3 is again tested. The procedure is executed for each new value of identifier-5 until condition-3 is found to be true. At this point identifier-5 will have assumed various values for one particular value of identifier-2 and the procedure will have been executed once for each value. Identifier-5 is reset to its initial value, identifier-2 is augmented in value, and condition-2 is again evaluated. If condition-2 is still false, the procedure is again executed for each value of identifier-5 until condition-3 is true. This process continues until condition-2 is found to be true following an augment in the value of identifier-2 after condition-3 is found to be true.

The PERFORM statement is complete when the specified procedure has been executed for each combination of the various values assumed by the identifier-2 and the identifier-5; control transfers to the statement following the PERFORM statement.

The following diagram shows how the PERFORM statement functions when three identifiers are to be varied. Operation is the same as above, except that a third level is added. The three items identifier-2, identifier-5, and identifier-8 are first set to initial values. The value of identifier-8 goes through a complete cycle for each value of identifier-5 which in turn goes through a cycle for each value of identifier-2.



Regardless of the number of identifiers in the VARYING option, the PERFORM statement is completed as soon as condition-2 is found to be true. Condition-2 is evaluated before the procedure is executed for the first time and if it is true, the procedure will not be executed even though other conditions (-3, -4) may not be true. After completion of the PERFORM statement, the value of identifier-2 differs from its last-used value by one increment (or decrement). Identifier-5 and -8 have their initial values. The following rules also apply:

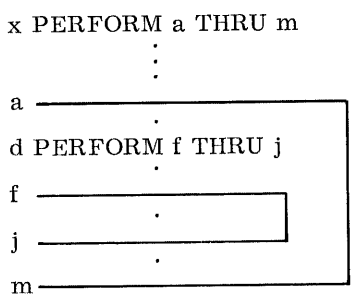
1. The increment (as specified following the word BY) should be a non-zero integer. It may be negative, however, making it a decrement.
2. Identifier-2, -5, and -8 should not refer to the same item; they should not be alternative names for the same data item.

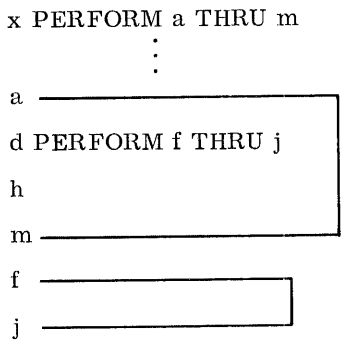
The following discussions apply to all four options of the PERFORM statement:

After execution of the procedure under control of the PERFORM statement, control returns to the statement following the PERFORM statement. The procedure to be executed by a PERFORM statement should not appear directly following that statement. If it does, it will probably be executed one more time than intended.

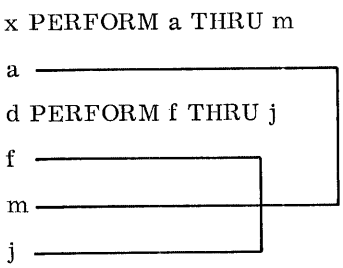
If a procedure referenced by a PERFORM statement includes within it another PERFORM statement, the procedure associated with the second PERFORM must be either entirely included in, or entirely excluded from, the procedure referenced by the first PERFORM.

Correct Specifications



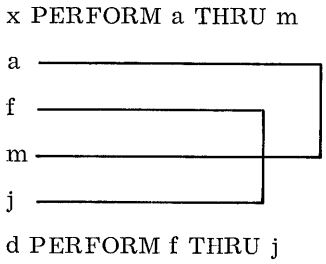


Incorrect Specifications

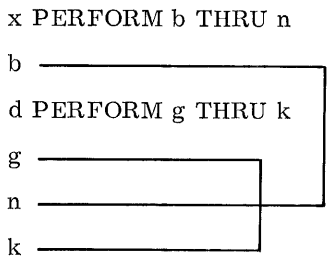


A procedure associated with one PERFORM statement can overlap or intersect the procedure associated with another PERFORM statement, if neither procedure includes the PERFORM statement associated with the other procedure.

Correct



Incorrect



Examples:

```
PERFORM SUMMARY.

PERFORM CALCULATION 4 TIMES.

PERFORM ISSUE NO-COPIES TIMES.

PERFORM REORDER UNTIL ON-ORDER + SUPPLY = AV-USE * 2 OR SUPPLY
    GR AV-USE * 2.

PERFORM RATE-CALC VARYING QUAN FROM 50 BY 5 UNTIL QUAN GR 200.
```

This statement stores one logical record from an input file in the record area in memory associated with that file.

Format 1: (used to read sequential files from any device)

READ file-name-1 RECORD [INTO identifier-1] AT END any-imperative-statement

Format 2: (used to read random-access mass storage files only)

READ file-name-2 RECORD [INTO identifier-2] INVALID KEY any-imperative-statement

An OPEN statement for the file must be executed prior to the first READ statement. The formats differ in the method used to determine the limits of the file. The first format is used to read all tape files and any mass storage files specified as sequential access; the AT END statement is executed when an end of file mark is read. The second format reads mass storage files specified as random access; the INVALID KEY statement is executed when the limit as specified in the FILE-LIMIT clause of the Environment Division is exceeded, or when the ACTUAL or SYMBOLIC KEY indicates that the requested record is not present.

When any READ statement is executed, the next logical record in the file is stored in the record area assigned to the file. The stored record remains in the record area until the next READ statement stores the next record in the area and overwrites some or all of the first record. Therefore if two records are required concurrently, the first must be moved from the record area before the second is read.

When the INTO option is used, the logical record is stored in both the record area and the area in memory specified by the identifier. If the format of this area differs from that of the record, the record is stored in accordance with the rules for a simple MOVE without CORRESPONDING. The INTO option makes one logical record available in both the record area and in an area in memory; of if the next logical record is read into the record area, two consecutive logical records are available concurrently.

The AT END option or the INVALID KEY option must be included with every READ statement specified in a COBOL source program. These options indicate, by their associated imperative statements, the action to be taken when the last record of a file has been read, or when a record with an invalid key is requested. Any attempt to execute a READ statement after an AT END or INVALID KEY option has been executed is an error unless a subsequent CLOSE and OPEN have been executed for the same file.

When a READ statement is executed for a random access mass storage file, the current value of data-name-5 in the ACTUAL/SYMBOLIC KEY clause determines the location of the next available record; except if a SEEK is executed prior to the READ, the value of data-name-5 at the time of SEEK determines the location of the next record. The contents of data-name-5 must be set by the user prior to the SEEK or prior to READ if SEEK is omitted. If the key is ACTUAL, data-name-5 is a numeric integer index to the position of the record in the file index; if the key is SYMBOLIC, data-name-5 is a seven-character name assigned to the record in the file index. If the contents of

the key are invalid or if the file limits are exceeded, the statement associated with the INVALID KEY clause is executed.

It is the user's responsibility to determine which condition caused execution of the INVALID KEY statement. The following example shows how this might be done.

Examples:

1.
```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    .
    .
    .
    FILE-LIMIT IS 20.
    SYMBOLIC KEY IS KEY-FILEA.
    .
    .
    .
DATA DIVISION.
WORKING-STORAGE SECTION.
77 KEY-FILEA PICTURE X(7).
    .
    .
    .
PROCEDURE DIVISION.
    .
    .
    .
    IF COUNT NOT GR FILE-LIMIT ADD 1 TO COUNT
        ELSE GO TO REC-OUT-OF-RANGE.
    MOVE "BLIVET" TO KEY-FILEA.
    READ FILEA INVALID KEY GO TO NON-EXIST-REC.
```

In the above example, the procedure REC-OUT-OF-RANGE will be executed when the file limit is exceeded, and NON-EXIST-REC will be executed when the contents of the key are invalid for any other reason. This could occur if a record is requested that is within the file limits as specified by FILE LIMIT but which was not generated on the file by a previous WRITE.

2.
```
READ MASTER-FILE AT END MOVE 1 TO MASTER-ENDED-IND
GO TO END-MASTER.
```

3.
```
READ DETAIL-FILE AT END GO TO END-DETAIL.
```

4.
```
READ INVENT-FILE AT END ADD 1 TO INVENT-IND GO TO END-INVENT.
```

5.
```
READ DATA-FILE INTO ITEM-1 INVALID KEY DISPLAY  "END DATA".
```

The RELEASE statement transfers records to the initial phase of a sort operation.

RELEASE record-name-1 [FROM identifier-1]

The RELEASE statement applied to a sort file is equivalent to a WRITE statement. Record-name-1 must be named in the DATA RECORDS clause of a sort file description entry (SD). Identifier-1 must be a Working-Storage area or an input record area, and a different data item than record-name-1.

RELEASE causes the record named by record-name-1 to be released to the initial phase of a sort. If the FROM option is included, the contents of the data area identified by identifier-1 are moved to record-name-1 and then released to the sort file. The move is made according to the rules for a simple MOVE. After a RELEASE statement is executed, the contents of the record area named by record-name-1 are no longer available; however, the contents of the data area identified in the FROM option will be available if that option is specified.

RELEASE must be part of an input procedure in the SORT statement (see SORT). When control passes from the input procedure, the file consists of all those records placed in it by execution of RELEASE statements.

RETURN

The RETURN statement obtains the sorted records from the final phase of a SORT operation.

RETURN file-name-1 RECORD [INTO identifier-1] AT END any-imperative-statement

RETURN is equivalent to a READ statement for a sorted file. It may be used only within the range of an ouput procedure associated with a SORT statement for file-name-1 (see SORT). File-name-1 must be described in a sort file description entry (SD). The identifier-1 must be a Working-Storage area or an output record area.

Execution of the RETURN statement causes the next record, in the order specified by the ASCENDING/DESCENDING key, to be made available for processing in the record area associated with the sort file. The move is made according to the rules for a simple MOVE.

If INTO is specified, the data is available in both the input record area and the data area associated with identifier-1. The INTO option may be used only when the input file contains just one type of record.

After execution of the imperative statement specified in the AT END phrase, no RETURN statement may be executed within the current output procedure.

The SEEK statement initiates record accessing on a random access mass storage file for subsequent reading or writing.

SEEK file-name-1 RECORD

SEEK uses the contents of data-name-5 in the ACTUAL/SYMBOLIC KEY clause in the FILE-CONTROL paragraph of the Environment Division for the location of the record. At the time of execution, the validity of the contents of data-name-5 is determined. If the key is invalid, the program will execute the imperative statement in the INVALID KEY clause of the next executed READ or WRITE for the associated file. The contents of data-name-5 are the responsibility of the user.

Example:

```
    ACTUAL KEY IS MAST-FILE-KEY.
        :
77  MAST-FILE-KEY PICTURE 9(3) USAGE IS COMPUTATIONAL.
        :
        :
    ADD 1 TO MAST-FILE-KEY.
    SEEK MASTER-FILE.
    READ MASTER-FILE INVALID KEY GO TO INVALID-PROC.
```

Contents of MAST-FILE-KEY at SEEK execution are a numeric index to the location of the desired record in the file index. If contents of the key are invalid, no record is read and control transfers to INVALID-PROC.

```
SORT file-name-1 ON { DESCENDING } KEY identifier-1 [identifier-2]...
                    { ASCENDING  }

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

The SORT statement creates a sort-file by executing input procedures or by transferring records from another file, sorts the records in the sort-file on a set of specified keys, and in the final phase of the sort operation, makes available each record from the sort-file, in sorted order, to some output procedures or to an output file.

The ASCENDING clause specifies a sorting sequence from lowest to highest value of KEY; DESCENDING specifies a highest to lowest sequence. The order of values is specified in the COBOL collating sequence in Appendix B.

File-name-1 is the sort file which must be described in a sort file description entry (SD). Each identifier must represent data items described in records associated with file-name-1. Section-name-1 is the name of an input procedure; section-name-2 is the name of an output procedure. File-names-2 and -3 must be described in the file description entries (FD); they are not sort files and must not be described in an SD entry. Identifiers following the word KEY are listed from left to right in decreasing significance without regard to how they are divided into KEY clauses. The KEY identifiers may not be variable length items, and cannot contain or be subordinate to entries that contain an OCCURS clause. If KEY identifiers appear in more than one record, the data descriptions must be equivalent and must start in the same relative position in each record.

An INPUT PROCEDURE must contain at least one RELEASE, and an OUTPUT PROCEDURE must contain at least one RETURN. They cannot contain SORT statements, nor may they transfer control outside the procedure or be transferred into from another part of the Procedure Division.

SORT is equivalent to the following operations when the INPUT and OUTPUT PROCEDURES are used:

- Opens sort file.
- Executes the INPUT PROCEDURE which prepares records and writes them into the sort file by executing RELEASE statements.
- Records collected by the RELEASE statement are sorted when the program allows control to go to the end of the INPUT PROCEDURE. The sort file is then closed, the data sorted, and the sort file is opened for input which takes place in the OUTPUT PROCEDURE.
- Control is given to the OUTPUT PROCEDURE and a record is read from the sort file each time a RETURN statement is executed.

- When the end of the sorted file is reached, the AT END statement of the last RETURN statement executed within the OUTPUT PROCEDURE is executed.
- When control is allowed to go to the end of the OUTPUT PROCEDURE, the sorting operation is terminated even if all the data in the sorted file has not been read. Control then proceeds to the statement following the SORT statement.

If the USING option 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 automatically performs the function of the OPEN, READ, USE, and CLOSE statements for file-name-2.

If the GIVING option is used, 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 is automatically opened before transferring the records and closed after the last record in the sort-file is returned.

STOP

This verb specifies a temporary or final halt in the execution of the object program.

$$\underline{\text{STOP}} \left\{ \begin{matrix} \text{literal-1} \\ \underline{\text{RUN}} \end{matrix} \right\} .$$

If a literal is specified, the object program displays the literal and halts temporarily. The word RUN after the word STOP produces an end-of-program exit to the supervisory control system. This option terminates program execution. A STOP RUN statement should be used only as the final statement of a sequence. A program will compile correctly without a STOP RUN statement; however, execution will not terminate normally.

To continue after a halt, the operator must type in X.GO for interpretation by the SCOPE system. X indicates the control point number 1-7 under which a program is running.

This verb subtracts one or the sum of more than one numeric value from a specified value and sets the result as the value of one or more items.

Format 1:

```
SUBTRACT {identifier-1} [{identifier-2}] ... FROM identifier-m [ROUNDED]
         {literal-1   }  [{literal-2   }]

[identifier-n [ROUNDED]] ... [ON SIZE ERROR any-imperative-statement].
```

Format 2:

```
SUBTRACT {identifier-3} [{identifier-4}] ... FROM {identifier-m} GIVING identifier-n
         {literal-3   }  [{literal-4   }]      {literal-m   }

[ROUNDED] [identifier-o [ROUNDED]]... [ON SIZE ERROR any-imperative-statement.]
```

Format 3:

```
SUBTRACT CORRESPONDING identifier-5 FROM identifier-6 [ROUNDED]

[identifier-7 [ROUNDED]] ... [ON SIZE ERROR any-imperative-statement].
```

In the first format one or more subtractions may be specified by a single statement, with the results as the values of different items. The literals and/or the values of the items named preceding FROM are summed and the total is subtracted from the value of the identifier-m item; the difference is set as the value of the identifier-m item. The same total is subtracted from the value of each item named after identifier-m (such as identifier-n) and the difference in each case is set as the value of the named item.

The second format stores the result of a subtraction as the value of one or more items. The literals and/or the values of the items named preceding FROM are summed and the total is subtracted from literal-m or from the value of the identifier-m, whichever is specified. The difference is set as the value of the identifier-n, and also as the value of all other items named after identifier-n (such as identifier-o).

In both formats 1 and 2, all items must be described in the Data Division as elementary numeric items. If a name encountered during compilation is not an elementary item, the processor issues a diagnostic and compilation of the SUBTRACT statement terminates. All literals specified must be numeric.

SUBTRACT CORRESPONDING

The values of one or more elementary items within a group are subtracted from the values of one or more selected elementary items within other groups. No literals may be included and all identifiers must be the names of groups. The groups are considered in pairs for the process of selecting one or more items from within each group for subtraction. The group named identifier-5 is paired in turn with each of the other groups named. For any pair of groups, the value of the elementary item selected from the identifier-5 group is subtracted from the value of the corresponding elementary item from the second-named group and the difference is set as the value of the item in the second-named group.

A pair of elementary items from a pair of groups is selected for subtraction if:

a. Their names are the same

b. Names of all higher-level items in each group (qualifiers for each item), up to but not including the names of the groups, are identical

c. Both items are elementary

After a pair of items has been selected for subtraction, the operation is the same as if a SUBTRACT statement of the first option had been specified. The same rules apply to subtraction of selected items as to subtraction of items specified in format 1.

After all matched pairs have been subtracted for identifier-7 and identifier-6 groups, the matched pairs are subtracted for identifier-5 and identifier-7 groups, and so on.

The following rules apply to subtract corresponding only:

1. No item described by an OCCURS clause in the Data Division can be involved in subtraction.
2. Within a specified group, any item with a REDEFINES clause is ignored, as are all items within the redefinition.
3. Data items with level number 77 (independent working storage or constant items) cannot be referenced.
4. Within a specified group, any item with A RENAMES clause in the Data Division entry is ignored.

Example:

```
SUBTRACT SOC-SEC FEDERAL-TAX TOTAL-DEDUCTIONS FROM MONTHLY-GROSS
GIVING MONTHLY-NET.

SUBTRACT CORRESPONDING UPDATE-RATE-TABLE FROM RATE-TABLE.

SUBTRACT DAY OF CURRENT-DATE FROM 30 GIVING DAYS-LEFT ON SIZE ERROR ADD 1
TO ERROR-COUNTER.
```

This verb introduces procedures to be executed in addition to the standard input-output error and label record handling procedures, and specifies Procedure Division statements to be executed just prior to producing a report group named in the Report section. The USE statement must be a sentence immediately following the section header for the declarative procedures which it introduces. The remainder of the section consists of paragraphs which define the operations.

SORT files must not be specified in a USE statement.

Format 1:

```
USE AFTER STANDARD ERROR PROCEDURE ON { file-name      }
                                      { INPUT          }
                                      { OUTPUT         }
                                      { INPUT-OUTPUT   }
                                      { I-O            }
```

The procedures included with a USE Sentence of format-1 are executed after the standard input-output error procedures of the input-output control system.

Format 2:

```
USE { BEFORE } STANDARD [{ BEGINNING }] [{ REEL }]
    { AFTER  }         [{ ENDING    }]  [{ FILE }]

LABEL { PROCEDURES } ON { file-name-2    }
      { PROCEDURE  }    { INPUT          }
                        { OUTPUT         }
                        { INPUT-OUTPUT   }
                        { I-O            }
```

The USE sentence in format 2 introduces the procedures to be executed in conjunction with those provided by the operating system. The procedures will be executed before or after input labels are checked, before output labels are prepared, or after output labels are written on tape. If label records are specified as STANDARD, the procedures are performed in addition to the label checking and writing handled by the operation system. If neither BEGINNING nor ENDING is specified, the procedures are executed for both beginning and ending labels. If neither REEL nor FILE is specified, the procedure is executed for both reel and file labels.

Format 3:

```
USE BEFORE REPORTING identifier-1 [identifier-2]...
```

Format 3 is used only on conjunction with the Report Writer. It is described in Chapter 5 with the Report Writer statements.

USE with Standard Labels

The following table shows the effect of changes made by USE procedures on the label area of the File Environment Table. This table applies only to USE procedures with standard label processing.

USE Statement Options			File Type	Label Area Use
BEFORE	BEGINNING	FILE REEL	OUTPUT	Changes will be reflected in opening labels.
		FILE REEL	INPUT	Changes will effect checking of opening labels.
AFTER	BEGINNING	FILE REEL	OUTPUT	Changes will have no effect
		FILE REEL	INPUT	Actual label information from beginning labels is available.
BEFORE	ENDING	REEL FILE	OUTPUT	Changes will be reflected in ending label.
		REEL FILE	INPUT	Changes effect checking of label.
AFTER	ENDING	FILE REEL	OUTPUT	Changes have no effect.
		FILE REEL	INPUT	No information available here.
Immediately after a file has been opened for input when any USE...REEL... statement applies to this file.				Label information from beginning labels still available.

USE With Non-Standard Labels

USE procedures have less meaning for label records with the data-name option (non-standard labels). When this option is used, the first record of the file is assumed to be a data-name-7 type record. It is read or written when OPEN occurs without a READ or WRITE in the COBOL source program. It can be referenced in a manner similar to all data records and is placed in the same area. Effectively, an OPEN INPUT corresponds to an OPEN INPUT and a READ, and OPEN OUTPUT corresponds to an OPEN OUTPUT and a WRITE data-name-7. USE procedures may be used, but are not the only way a user has access to the label record.

Non-standard end-of-reel labels on input files may be checked only if the VALUE OF ENDING-TAPE-LABEL-IDENTIFIER clause is specified in the label records clause. This identifier is used to distinguish between end-of-file and end-of-reel labels. If this option is used, USE procedures referring to reels may be used for input tapes with non-standard labels. The results of USE procedures for such files are outlined in detail in section 6.1.5.

This statement releases one logical record from the output record area to the input-output control system; the system then writes the record on an external device.

Format 1:

```
WRITE record-name-1 [FROM identifier-1] [ { BEFORE } ADVANCING
                                        { AFTER  }
                                                  { { integer      } }
                                                  { { identifier-2 } } LINES ]
                                                  { { mnemonic-name} }
```

Format 2:

WRITE record-name-2 [FROM identifier-2] INVALID KEY any-imperative-statement

An OPEN statement giving the name of the file containing the record to be written must be specified before a WRITE statement may be executed.

Format 1 of the WRITE statement is used with all files; Format 2 is used with mass storage files. The statement associated with INVALID KEY is executed when the key in the ACTUAL or SYMBOLIC KEY clause is found invalid. This key is the contents of data-name-5 in the ACTUAL/ SYMBOLIC KEY clause and must be a numeric index to the location of the record (ACTUAL KEY) or a seven-character name identifying the record in the name index (SYMBOLIC KEY). A key is invalid when limits specified in the FILE LIMITS clause of the Environment Division are exceeded or when the key specifies a non-existent record.

The FROM option is used to move a record from the area in memory specified by the identifier to the output record area and simultaneously release this record to the input-output control system. The record is moved in accordance with the rules for a simple MOVE. The name of the record may not be the same as the identifier in the FROM option.

The ADVANCING option controls positioning of each record on a printed page; it is used only with the printer. When LINES is used, the printer spaces the number of lines specified by the integer or identifier. The integer must be positive and the identifier must result in a positive integer at object time. The mnemonic-name is always associated with a carriage control character. The particular character must be specified in the non-numeric-literal IS mnemonic-name clause of the Special-Names paragraph in the Environment Division. Even if a carriage control character is not specified, the user must always reserve a character position for the carriage control in any record to be printed. The legal control characters are:

(blank)	space 1 line	1	page eject
0	space 2 lines	+	no advance

Any character other than those specified above has the same meaning as blank when used in the carriage control position. No negative spacing is permitted, and spacing in excess of 66 lines results in ejection to a new page.

WRITE statements may be mixed with WRITE BEFORE/AFTER ADVANCING statements. The WRITE with ADVANCING option stores the spacing information in the first character position of the record. It replaces any contents of that position previously supplied by the user. WRITE without ADVANCING will not disturb this first character position.

Mixed use of WRITE, WRITE BEFORE, and WRITE AFTER ADVANCING will result in the spacing shown in the table below:

Previous Statement \ New Statement	WRITE	WRITE BEFORE c	WRITE AFTER d
WRITE	PRINT LINE 1. Space according to char 1 of line 1. PRINT LINE 2.	PRINT LINE 1. Space according to char 1 of line 1. PRINT LINE 2.	PRINT LINE 1. Space according to char 1 of line 1 followed by d spaces. PRINT LINE 2.
WRITE BEFORE a	PRINT LINE 1. a spaces. PRINT LINE 2.	PRINT LINE 1. a spaces. PRINT LINE 2.	PRINT LINE 1. a + d spaces. PRINT LINE 2.
WRITE AFTER b	Overprint.	Overprint.	PRINT LINE 1. d spaces. PRINT LINE 2.

If the ADVANCING option is not used, and if the records are to be printed, the first character of each record must contain a print control character.

Examples:

```
WRITE MASTER-REC.

WRITE DETAIL OF MASTER-OUT FROM DETAIL OF WORK-AREA.

WRITE PRINT-LINE AFTER ADVANCING 2 LINES.

WRITE HEADER-LINE BEFORE ADVANCING 3.

WRITE ERROR-REC INVALID KEY ADD 1 TO ERR-COUNT GO TO ERROR-PROC.
```

THE REPORT WRITER 5

The Report Writer enables the user to specify the format of printed reports to be output from the COBOL program. Each report is defined in the Report Section of the Data Division using the formats described in this chapter. Once a report is defined, the Report Writer statements in the Procedure Division place the report in the specified format on an intermediate storage device for printing. More than one report can be generated from a single source program.

When the Report Writer is used, the Report Section must be included as the last section of the Data Division, and File Description entries (FD) in the File Section must contain the names of the reports to be output.

The Report Writer will be implemented first in version 2.0 of 6400/6500/6600 COBOL.

5.1 GENERAL DESCRIPTION

A report is a pictorial presentation of data. In preparing a report, the format is differentiated from the content. The format must be planned in terms of page width and length, organization of report items on the page, and the hardware device on which the report will be written.

Each report is divided into report groups, which may be further divided into group items and elementary items. A report group is a set of related data within a report. Any group or item within a report group which contains subordinate items is referred to as a group item. An item that contains no subordinate items is an elementary item.

The concept of levels is used in organizing a report. The highest level is the report itself, level indicator RD followed by the report name. Next are report groups at 01 level. They may contain lower level group and/or elementary items with level numbers from 2-49. Each group at the 01 level is defined according to type. The type may be a heading group, footing group, control group, or detail group. A report group may extend over several lines of a page.

The user may refer from the Procedure Division to the report-name and to 01 level detail report groups; he may refer from the Declaratives Section to other 01 level report groups. He may refer from the Declaratives Section or the Procedure Division to any elementary item that is named and contains a SUM clause.

Every report description must contain a report description entry, and a report group description entry. The report description entry (RD) specifies the overall format: characteristics of the page are outlined; limits are prescribed for the page and for footings, headings, and detail information within the page. This entry also specifies data items that act as control factors during printing. Each report associated with an output file must be defined in an RD entry.

Each report group is a set of one or more data items; it may consist of one or several lines. A report group must contain an 01 level number and a TYPE clause. The data-name is optional; however, it must be specified with any report group, group item, or elementary item description if it is referred to by GENERATE or USE statements in the Procedure Division. The order in which report groups are specified is important only if subcompiled programs refer to the same report. Otherwise the position of a group in the report is determined by the TYPE clause. Any elementary item descriptions must be written in the order (left to right) in which the items they describe are to appear on the printed page.

5.1.1 CONTROL GROUPS

Summary information may be presented within the body of a report. The concept of a control hierarchy makes it possible to automatically produce required summary information together with any heading, detail and footing information in a control group. Control items are specified in the report description in the same order as the control hierarchy. Any change in the contents of a control item produces a control break. Changes are recognized between executions of GENERATE statements and they set in motion the automatic production of control footing and heading report groups associated with the item. The set of control heading, control footing, and associated detail report groups constitute a control group for a given control data-name. Within the hierarchy, lower level heading and/or footing report groups are included in a higher level control group.

5.1.2 PAGE/OVERFLOW CONDITIONS

Page and overflow heading and footing report groups are mutually exclusive, and are effective only when the condition occurs. When, following the rules associated with "last detail line," a control group is completed on one page and the next control group is to be started on the next page, then a "page condition" exists. If a page is completed any other way, an overflow condition exists. The page footing group is printed following the last detail or control footing group if the page condition exists, else the overflow footing is printed. If only one overflow specification is given, either footing or page, both are assumed.

If neither page nor overflow headings and/or footings are specified, none are printed when a page break occurs.

5.1.3 SPECIAL REGISTERS

The fixed data names, LINE-COUNTER and PAGE-COUNTER are generated automatically by the Report Writer according to specific entries in a report description.

LINE-COUNTER is used by the Report Writer to determine when a page/overflow heading and/or footing report group is to be presented, and to control spacing of information on the page. One line counter is supplied for each report described in the Report Section; it is a numeric item set to a fixed size by the compiler. The line counter may be referred to by Procedure Division statements; however, if it is changed with a Procedure Division statement, page format control may be unpredictable. If more than one line counter exists because more than one report name (RD) entry is

specified, the line counter may be qualified by the report name. Initially, the line counter is set to zero by the Report Writer. It is automatically tested and incremented during execution according to the PAGE LIMIT clause and the values specified by LINE NUMBER and NEXT GROUP. The line counter is reset to zero when the page limit is exceeded during execution. The line counter is also reset to zero when a page break occurs. After generation of any specified report groups, the line counter is set to the value of the absolute LINE NUMBER or absolute NEXT GROUP, if this value is less than or equal to the contents of the line counter at that time.

During execution, the value of the line counter represents the number of the last line of the previous report group, or it represents the number of the last line skipped by NEXT GROUP specification. This value of the line counter is tested from the Procedure Division.

PAGE-COUNTER is a special register automatically generated by the Report Writer as a data item to number the pages within a report. One page counter item is supplied for each report. It is a numeric item set to a fixed size by the compiler.

The page counter may be referenced by the Procedure Division statements. Since the Report Section may contain more than one report description entry, the user may qualify the page counter by the report name to make it unique.

The page counter is automatically set to one when the Report Writer begins a report. With a Procedure Division statement immediately following the INITIATE statement, the user may set the page counter to an initial value greater than one. The Report Writer automatically increments the page counter by one at each page break. It is incremented after any page or overflow footing and before any page or overflow heading.

5.2 DATA DIVISION ENTRY FORMATS

Formats used by the Report Writer are specified below in the order of appearance in a COBOL program. The report must be named in the File Section and described in the Report Section of the Data Division. Statements which generate a report are specified in the Procedure Division.

File Section

An output file used by the Report Writer must be described in a File Description Entry in the Data Division. The FD entry must name the reports in the REPORT IS clause. No data records named by a DATA RECORD clause may be included in the same FD entry. The format of the File Description Entry is described in Chapter 3.

$$\left\{\begin{matrix}\underline{\text{REPORTS}}\ \text{ARE}\\ \underline{\text{REPORT}}\ \text{IS}\end{matrix}\right\}\ \text{report-name-1 [report-name-2]...}$$

A report name may be specified in exactly the same way as a data name. However, if subcompile is used, the report name must be unique in the first six characters, must begin with an alphabetic character, and may not contain hyphens. If more than one report name is included in an FD entry, the file will contain more than one report. If separate files are required for each report, an FD entry must be specified for each file.

Report Section

Each report named in an FD entry must be defined in a report description entry in the Report Section which must be the last section in the Data Division. The header REPORT SECTION followed by a period must precede any report descriptions, and the header must be alone on a line. This section specifies the layout of each page. The report description entry (level indicator RD) is required, and report group description entries (level number 01) are required to divide the report into groups.

5.3 REPORT DESCRIPTION ENTRY

The physical structure and overall format of a report is specified as follows:

Format 1:

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

This format is used only when the complete RD entry is contained in the COBOL library.

Format 2:

RD report-name-2 [WITH CODE mnemonic-name-2]

$$\left[\left\{\begin{matrix}\underline{\text{CONTROL}}\ \text{IS}\\ \underline{\text{CONTROLS}}\ \text{ARE}\end{matrix}\right\}\ \left\{\begin{matrix}\underline{\text{FINAL}}\\ \text{data-name-1 [data-name-2]...}\\ \underline{\text{FINAL}}\ \text{data-name-3 [data-name-4]...}\end{matrix}\right\}\right]$$

$$\left[\underline{\text{PAGE}}\ \left\{\begin{matrix}\underline{\text{LIMIT}}\ \text{IS}\\ \underline{\text{LIMITS}}\ \text{ARE}\end{matrix}\right\}\ \text{integer-1}\ \left\{\begin{matrix}\underline{\text{LINE}}\\ \underline{\text{LINES}}\end{matrix}\right\}\right.$$

$$\left.\begin{matrix}[\underline{\text{HEADING}}\ \text{integer-2}]\\ [\underline{\text{FIRST DETAIL}}\ \text{integer-3}]\\ [\underline{\text{LAST DETAIL}}\ \text{integer-4}]\\ [\underline{\text{FOOTING}}\ \text{integer-5}]\end{matrix}\right]$$

The RD level indicator is required; it starts in column 8 and precedes the report name. The unique report name must be specified here and in an FD entry in the File Section.

All clauses following report name in this entry are optional.

CODE is used when more than one report will be generated and stored on the same intermediate file for subsequent printing. It specifies a unique character that identifies this report.

WITH CODE mnemonic-name-1

This clause immediately follows the report name in the RD entry. The mnemonic-name must be equated to a non-numeric literal in the Special-Names paragraph of the Environment Division. The literal must be one character (in quotation marks) and it may not itself be a quotation mark.

This is a unique identifier so that, following execution, a report selection program can inspect a file and print only the reports required. Report selection programs must be supplied by the user.

CONTROL

This clause specifies data-names associated with the hierarchy of control within a report.

$$\left\{\begin{matrix}\text{CONTROL IS}\\ \text{CONTROLS ARE}\end{matrix}\right\} \left\{\begin{matrix}\text{FINAL}\\ \text{data-name-1 [data-name-2]...}\\ \text{FINAL data-name-1 [data-name-2]...}\end{matrix}\right\}$$

Since the data names indicate the control hierarchy, they must be listed in order from major to minor. FINAL is the highest control, data-name-1 is the major control, data-name-2 is next in order, and the last name is the minor control. The data names must be defined in the File, Working-Storage, or Common-Storage Section of the Data Division.

This clause is optional; however, it must be included when the TYPE clause specifies control footing or control heading. Control footings and headings are printed automatically as a result of control breaks defined in this clause. A control break occurs whenever the value of a data-name specified in this clause changes. The CONTROL clause must also be included when the RESET clause is specified in a report element description to reset the sum counters associated with data names listed with CONTROL.

This clause is required if PAGE HEADING, PAGE FOOTING, OVERFLOW HEADING, or OVERFLOW FOOTING are specified in the TYPE clause or if LINE NUMBER or NEXT GROUP is specified for an item. The PAGE LIMIT clause may be omitted if no automatic positioning of report groups on the page is desired.

Only one PAGE LIMIT clause may be specified for each RD entry, it gives specific line control for positioning reports on a page. All the integers must be positive numbers.

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

Integer-1 indicates the maximum lines to be printed on a page. It must be equal to or greater than integer-5 which specifies the last line on which a control footing group can appear.

HEADING specifies the number of the first line on which the heading can appear; it must be greater than or equal to 1. If HEADING is not specified, it is assigned a value of 1.

FIRST DETAIL specifies the first line on which a detail report group can start. If a heading extends beyond the line specified by integer-3, the detail group will follow the last heading line. Integer-3 must be equal to or greater than integer-2. If FIRST DETAIL is omitted, it is assigned the value of integer-2. The detail line can begin on the first line following the heading; or, if no heading is specified, on the first line.

LAST DETAIL specifies the last line on which a detail group report can be printed. Integer-4 must not be less than integer-3. If LAST DETAIL is not specified it is assigned the value of integer-5. The last line on which detail information can appear is either the last line on which a control footing group can appear (integer-5); or, if no footing is specified, the last line of detail information may be the same as the last line of the page (integer-1).

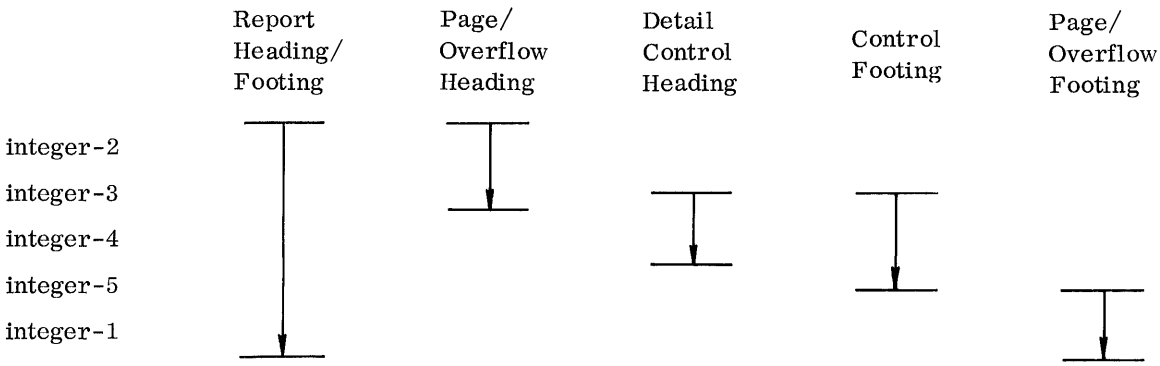
FOOTING specifies the number of the last line on which a control footing group can appear. It must be equal to or greater than integer-4. If not specified, it is assigned the value of integer-4; if neither footing nor last detail is specified, both the integers are set to the value of integer-1. No control footing may begin before the first line of detail information (integer-3) or extend beyond the line specified by integer-5. Page and overflow footings may start with the line specified by integer-5, but they must not extend beyond the last line on the page (integer-1). A diagnostic is provided at compile time if page or overflow footings do not fit within the page limits specified.

If absolute line spacing is desired for all groups in the report, only the integer-1 LINES need be specified. In this case, no page or overflow footings or headings should be specified.

Example:

```
REPORT SECTION.
RD   RATIO-REPORT
     PAGE LIMIT IS 55 LINES.
```

The following chart represents the limits of page format when all options of the PAGE LIMIT clause are specified:



5.4 REPORT GROUP DESCRIPTION ENTRY

This entry defines the characteristics of each report group and of any group or elementary items within the report group.

Format 1:

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

Format 1 is used only when data-name-2 names an item in the COBOL library or in the Data Division.

Format 2:

```
01 [data-name-2]
    TYPE IS { REPORT HEADING                                          }
            { RH                                                      }
            { PAGE HEADING                                            }
            { PH                                                      }
            { OVERFLOW HEADING                                        }
            { OH                                                      }
            { {CONTROL HEADING} {data-name-2}                         }
            { {CH             } {FINAL      }                         }
            { DETAIL                                                  }
            { DE                                                      }
            { {CONTROL FOOTING} {data-name-3}                         }
            { {CF             } {FINAL      }                         }
            { OVERFLOW FOOTING                                        }
            { OV                                                      }
            { PAGE FOOTING                                            }
            { PF                                                      }
            { REPORT FOOTING                                          }
            { RF                                                      }
    [LINE NUMBER IS {integer-1      } ]
                    {PLUS integer-2 }
                    {NEXT PAGE      }
    [NEXT GROUP IS {integer-3      } ]
                   {PLUS integer-4 }
                   {NEXT PAGE      }
    [CLASS IS {ALPHABETIC   }
              {NUMERIC      }
              {ALPHANUMERIC }
              {AN           }
    [SIZE IS integer-5 {CHARACTERS} ]
                       {DIGITS    }
    [USAGE IS DISPLAY].
```

Format 2 indicates a report group; a report group is always at the 01 level and includes all items between this entry and the next level 01 entry. Data-name-2 is optional but it must be included if it is referenced from the Procedure Division. The TYPE clause is required for all report groups.

Format 3:

```
nn [data-name-3]
[COLUMN NUMBER IS integer-1]
[ LINE NUMBER IS { integer-2          } ]
                 { PLUS integer-3     }
                 { NEXT PAGE          }
[GROUP INDICATE]
{ [ SOURCE IS { SELECTED data-name-2 } ]                                  }
              { LINE-COUNTER         }                                    }
              { PAGE-COUNTER         }                                    }
{ [SUM data-name-3 [data-name-4]...[UPON data-name-5                      }
{ [VALUE IS literal-1]                                                    }
[ RESET ON { data-name-6 } ]
           { FINAL       }
[CLASS IS { ALPHABETIC   }
          { NUMERIC      }
          { ALPHANUMERIC }
          { AN           }
[ SIZE IS integer-4 { CHARACTERS } ]
                    { DIGITS     }
[USAGE IS DISPLAY]
[PICTURE IS character-string]
[ { ZERO SUPPRESS          } [LEAVING integer-5 PLACES] ] [BLANK WHEN ZERO]
  { CHECK PROTECT          }                            ]
  { FLOAT DOLLAR SIGN      }
  { FLOAT CURRENCY SIGN    }
[SIGNED]
[JUSTIFIED RIGHT]
[POINT LOCATION IS { LEFT  } integer-6 PLACES].
                   { RIGHT }
```

Format 3 defines a group or elementary item within a report group. If a report group consists of only one elementary item, format 3 may include the TYPE and NEXT GROUP clauses in order to specify the report group and elementary item in the same entry. In this case, level number nn must be 01.

If LINE NUMBER is not specified in format 2, it must be specified for the first item within the report group. If LINE NUMBER is specified for a group item, the entire group is presented on the line indicated. If LINE NUMBER is specified for an elementary item, all subsequent items are presented on the line indicated until a new line number is specified. The line number for an item at a lower level may not contradict that specified for an item at a higher level.

The level number (nn) can be 01-49; it must be specified. Level numbers in a report description imply the same system of hierarchy as level numbers in a record description (Chapter 3, Data Division). The special level numbers: 77, 66, and 88 are illegal in Report Writer entries. Data-name-3 need be specified only if the item it identifies is referenced in the Procedure Division or mentioned in the SUM clause of another report element description. The data-name of a report element need not be unique if it can be qualified by a higher level data-name to make it unique.

The following clauses have the same meaning for a report description entry and a record description entry; they are defined in Chapter 3:

COPY

CLASS

SIZE

USAGE

PICTURE and Editing Clauses

JUSTIFIED

SIGNED

POINT LOCATION

VALUE

Exceptions: USAGE must be specified as DISPLAY for all report items. CLASS is treated as ALPHANUMERIC for report items regardless of the specification. VALUE may not be used to specify a condition name.

Clauses peculiar to the Report Writer are described in the following pages; they appear in the order they are presented in the Report Group Description Entry formats:

TYPE

LINE NUMBER

NEXT GROUP

COLUMN NUMBER

GROUP INDICATE

SOURCE SUM-VALUE

RESET

This clause which determines the type of a report group, is required for every report group. A report usually consists of a title or heading for the whole report, headings and/or footings for each page, and at least one line or group containing the detail information, repeated as often as required. In addition, summary information can be printed at control breaks in the form of control headings and/or footings. Each type of report group must be specified in a TYPE clause.

```
         ( REPORT HEADING                                  )
         | RH                                              |
         | PAGE HEADING                                    |
         | PH                                              |
         | OVERFLOW HEADING                                |
         | OH                                              |
         |  {CONTROL HEADING}   {data-name-4}              |
         |  {CH             }   {FINAL      }              |
TYPE IS <  DETAIL                                           >
         | DE                                              |
         |  {CONTROL FOOTING}   {data-name-5}              |
         |  {CF             }   {FINAL      }              |
         | OVERFLOW FOOTING                                |
         | OV                                              |
         | PAGE FOOTING                                    |
         | PF                                              |
         | REPORT FOOTING                                  |
         ( RF                                              )
```

If type is DETAIL (DE), the GENERATE statement is required in the Procedure Division to produce the named report group. GENERATE will produce the specified page or overflow headings and/or footings, and any control headings and/or footings when control breaks occur. The INITIATE statement will produce any specified report heading group followed by any control heading final. The TERMINATE statement will produce the report footing if specified, as well as any control footing final.

REPORT HEADING/REPORT FOOTING

REPORT HEADING (RH) may be specified for only one report group; it is printed once at the beginning of the report. Similarly, REPORT FOOTING (RF) may be specified for only one group; it is printed once at the end of the report. An INITIATE statement is required to produce a report heading and a report footing. If SOURCE is specified in the description of a report footing, it refers to the value of the items at the time TERMINATE is executed. A SOURCE for a report heading group refers to the value of the items at the time INITIATE is executed. Nothing may precede the report heading or follow the report footing.

PAGE HEADING/PAGE FOOTING

PAGE HEADING (PH) is a report group printed at the beginning of each page. PAGE FOOTING (PF) is a report group printed at the end of each page following a page condition. A report may have only one page heading report group and one page footing report group.

OVERFLOW HEADING/OVERFLOW FOOTING

OVERFLOW HEADING (OH) is a report group printed at the beginning of a page following an overflow condition; OVERFLOW FOOTING (OV) is a report group printed at the end of a page following an overflow condition. Only one report group of either type may be specified for a report. The LAST DETAIL option of the PAGE LIMIT clause must be specified in the RD entry for this report if either an overflow heading or footing is specified.

A page or overflow heading group is printed according to the rules given for the page/overflow conditions. Similarly, the choice of a page or overflow footing is determined by page and overflow conditions. No one page may have both page and overflow headings or footings. If a report has no overflow groups, any page headings and footings specified are printed on every page regardless of the existence of an overflow condition.

CONTROL HEADING/CONTROL FOOTING

Any CONTROL FOOTING (CF) report group is printed following the control break specified by data-name-3 or any higher level control break; and any CONTROL HEADING (CH) report group is printed following the control break specified by data-name-2 or any higher level control break. Only one report group may be specified as a control heading for each data-name-2, and only one group may be specified as a control footing for each data-name-3.

CONTROL HEADING FINAL specifies a report group to be printed between the report heading and the first control heading group. Only one report group of this type is allowed in a report. A CONTROL FOOTING FINAL report group is printed between the end of a report and the report footing. Only one group of this type may be specified for a report.

If CONTROL HEADING or FOOTING is specified, the data names and/or FINAL must be specified in the CONTROL clause of the RD entry for the report. A control break must occur for either of these report groups to be printed.

A SOURCE clause in a control footing final report group refers to item values at the time the TERMINATE statement is executed.

DETAIL

A report group specified as DETAIL (DE) is printed each time a GENERATE statement referring to that group is executed. A detail type report group must have a unique data-name at the 01 level, which may be referenced by GENERATE. GENERATE will also print any applicable headings and footings automatically.

If all above report group types are specified, the Report Writer will print them in the following order:

REPORT HEADING (once only)
PAGE HEADING or OVERFLOW HEADING

CONTROL HEADING
DETAIL
CONTROL FOOTING

PAGE FOOTING or OVERFLOW FOOTING
REPORT FOOTING (once only)

CONTROL HEADING report groups are presented in the following order:

Final Control Heading
Major Control Heading
Minor Control Heading

CONTROL FOOTING report groups are presented in the following order:

Minor Control Footing
Major Control Footing
Final Control Footing

This clause is required for every report group. It indicates the line on which the report group is to be printed. It may be specified at the report group level or for the first elementary item in the group.

$$\underline{\text{LINE}}\ \text{NUMBER IS} \left\{ \begin{array}{l} \text{integer-1} \\ \underline{\text{PLUS}}\ \text{integer-2} \\ \underline{\text{NEXT}}\ \underline{\text{PAGE}} \end{array} \right\}$$

When LINE NUMBER is absolute, the line-counter register is set to the value of integer-1 and all items within the report group are printed at this line number until a new value for line-counter is specified. If integer-1 is equal to or less than the previously specified value of line-counter, a page or overflow condition exists and the report group is printed on the next page.

When LINE NUMBER is a relative value, the line-counter is incremented by integer-2 for this item and remains the same for subsequent items within the report group until a new LINE NUMBER clause resets the line-counter. When LINE NUMBER is specified for an elementary item, all subsequent elementary items appear on the same line until a new LINE NUMBER clause is encountered. LINE NUMBER for an elementary item may not contradict that specified for a group item. Within a report group, entries must be assigned line numbers in ascending order. Therefore, an absolute line number (integer-1) cannot be preceded by a relative line number (PLUS integer-2).

NEXT PAGE specifies the item that is the first to be printed on the following page. NEXT PAGE may be specified only at the 01 level.

Examples:

1)
```
01  GROUP A TYPE DETAIL LINE IS NEXT PAGE.
    03  PART-1 LINE 01 ...
    03  PART-2 LINE PLUS 01 ...
```

Each detail report group starts on a new page; the first entry in the group is on the first line, the second on the next line.

2)
```
01  TYPE RH LINE IS 1...
                NEXT GROUP IS PLUS 2.
01  TYPE DE LINE PLUS 1...
01  TYPE CF DATA-LIST  LINE PLUS 2...
```

The report heading group is printed on line 1; 3 lines are skipped before the first detail group is printed, each subsequent detail line is printed on the next line; 2 lines are skipped before the control footing group is printed.

This clause may appear only at the 01 level. It determines the spacing between report groups.

$$\underline{\text{NEXT GROUP}}\ \text{IS}\ \left\{\begin{array}{l}\text{integer-3}\\ \underline{\text{PLUS}}\ \text{integer-4}\\ \underline{\text{NEXT PAGE}}\end{array}\right\}$$

Integer-3 and integer-4 must be positive, and integer-3 may not exceed the maximum number of lines specified by the PAGE LIMIT clause.

The line counter is set to the value of integer-3 after the last line of the preceding report group is printed. This value indicates the line preceding the one on which the next report group is to be printed. If integer-3 is less than or equal to the previous value of the line-counter register, a page or overflow condition exists.

The line counter is incremented by the value of integer-4 which indicates the number of lines skipped before the next report group is printed.

NEXT PAGE produces an automatic skip to the next page following the last line of the current report group. Page or overflow headings and footings are produced as required.

Example:

```
01  DETAIL-LINE
    TYPE DETAIL
    LINE PLUS 01...
    NEXT GROUP PLUS 3.
```

DETAIL-LINE is printed on the line following the preceding group; and DETAIL-LINE will be followed by three blank lines before the next group is printed.

This clause indicates the absolute column number on the printed page of the leftmost character of the elementary item; it may be specified only at the elementary item level.

COLUMN NUMBER IS integer-1

This clause must be included in the description of every elementary item to be presented, otherwise, the elementary item is suppressed when the report is produced. Integer-1 must be positive. For a particular line, COLUMN NUMBER entries must be presented in the order in which the items are to appear on the page, from left to right. Column number 01 may not be used by the COLUMN NUMBER clause; it is reserved for spacing information supplied by the Report Writer.

When COLUMN NUMBER is specified, the elementary item description must also contain a PICTURE or SIZE clause and one of the clauses, SOURCE, SUM, or VALUE. However, if SOURCE IS SELECTED is specified at the group level and the elementary item is included in the selected group, only PICTURE or SIZE need be specified.

Example:

```
01  DETAIL-LINE
    TYPE DETAIL LINE PLUS 01.
    03 COLUMN 02 PICTURE X(10) SOURCE IS PROG-NAME.
    03 COLUMN 13 PICTURE ZZZZ9 SOURCE IS COBOL-LINES.
    03 COLUMN 20 PICTURE ZZZZ9 SOURCE IS GMAP-LINES.
    03 COLUMN 27 PICTURE Z9.99 SOURCE IS G-RATIO.
    03 COLUMN 34 PICTURE Z(5)9 SOURCE IS O-LINES.
    03 COLUMN 42 PICTURE ZZZ9.99 SOURCE IS O-RATIO.
```

This clause indicates the elementary item is to be produced only once at the first occurrence of the item following a control break or at the beginning of a new page. This clause may be specified only at the elementary item level within a DETAIL report group.

GROUP INDICATE

When GROUP INDICATE is specified, the elementary item is printed the first time the detail report group is printed following a control break. If a control break does not occur, this item is printed with the first detail report group containing the item on a new page. Regardless of the number of times the detail report group appears on a page, this item within the group only appears following a control break or at the first appearance of the group on a page.

Example:

```
01  DETAIL-ITEM TYPE DETAIL  LINE PLUS 1.
    02  DEPT PICTURE X COLUMN 20 SOURCE DEPT-NO
        GROUP INDICATE.
    02  SECT PICTURE X(3) COLUMN 30 SOURCE SECT-NO
        GROUP INDICATE.
    02  GRP-NO PICTURE  X(3) COLUMN 40 SOURCE GROUP-NO.
    02  AMOUNT PICTURE 9(5)  COLUMN 45 SOURCE AMNT-NO.
    .
    .
    .
```

In the above detail report group, the items called DEPT and SECT will only be printed the first time the detail group appears on a page and whenever a control break occurs. The other items in the group will be printed each time the detail group is printed.

The SOURCE, SUM, or VALUE clauses define the purpose of an item within the report group; only one may be included in the description of any one item.

```
{ SOURCE IS { LINE-COUNTER                    }                    }
{           { [SELECTED] data-name-8          }                    }
{           { PAGE-COUNTER                    }                    }
{ SUM data-name-9 [data-name-10]... [UPON data-name-11)            }
{ VALUE IS literal-1                                               }
```

A data-name indicates an item appearing in the File, Working-Storage, Common-Storage, or Constant Sections; or a data-name may be the name identifying a SUM counter in the Report Section. The literal specified in the VALUE clause may be numeric, non-numeric, or a figurative constant.

SOURCE without SELECTED, and the SUM and VALUE clauses may appear only at the elementary level; SOURCE IS SELECTED may appear only at the group level. SUM specifies an item to be summed, and may appear only with an item in a control footing report group.

SOURCE, SUM, or VALUE must be specified for every elementary item unless SOURCE IS SELECTED is specified for the group containing the elementary item.

SOURCE and SUM are described below; VALUE is described in Chapter 3, Data Division.

SOURCE

Indicates a data item to be used as the source for the report item. PICTURE or SIZE must also be specified in the entry for the report item. The value of the SOURCE data item is effectively moved to the report item and presented according to the PICTURE or SIZE specified in the report item description.

Data-name-8 is an item in the File, Working-Storage, Common-Storage, or Constant section whose value at object time is the effective value of the report item. If LINE-COUNTER is specified, the current value of the line-counter is the source; this value is the number of the last line printed or skipped. If SOURCE IS PAGE-COUNTER is specified, the current value of the page-counter is the source.

When SOURCE IS SELECTED, data-name-8 must be a group item. This option corresponds to a MOVE CORRESPONDING statement. The elementary level items within data-name-8 are matched against the data-names specified at the elementary level within the report group. Matching data items are selected as source items to be included and presented within the report group according to PICTURE or SIZE, CLASS and USAGE specifications given for the data items in the report group entry. The data names of the elementary items in the report group must be unique.

SUM indicates values to be accumulated when a control break occurs at object time; it may appear only in a control footing report group at the elementary level. Any item containing a SUM clause generates a binary accumulator used for summing. This accumulator (the SUM counter) may be referenced by using the data-name specified for the item containing the SUM clause. The data-names specified with SUM indicate items to be summed. They must appear as operands of a SOURCE clause, or they must name an item containing a SUM clause at an equal or lower level of control. Any lower level item that is the operand of a SUM clause at a higher control level must have a data-name as the subject of its entry. The data-names specified with SUM may be subscripted, and they must be qualified to make them unique. SIZE or PICTURE must be specified for each SUM counter; editing characters or clauses may be specified. Any editing occurs immediately before the contents of the SUM counters are printed. If editing is not specified, the SUM counter is treated as a numeric data item. When the SUM counter is referenced from the Procedure Division, a non-integral binary representation of the current contents of the SUM counter is returned.

The operands, specified in a SUM clause, are added to the SUM counter at each execution of a GENERATE statement unless SOURCE IS SELECTED was specified; in this case only the selected items containing SUM operands are summed.

The UPON option is used for selective summation. Data-name-9, -10, etc., must be SOURCE data items in data-name-11. Data-name-11 must be the name of a detail report group. The values of data-name-9, -10, etc., are added to the SUM counter only when data-name-11 is referenced by a GENERATE statement.

When GENERATE is executed for a detail report group, all summing takes place automatically as follows:

- The CONTROL data-name specified in the control footing report group is tested.
- If the data-name is unchanged, each SOURCE item in the DETAIL report group is added to each SUM counter in any higher level control footing report group which names the SOURCE item. Then the detail line is produced.
- If the data-name has changed, a control break has occurred. Each SUM counter in the lowest level report group is added to every SUM counter in the same report group that names the SUM counter as an operand. Then the control footing report group is produced.
- Each SUM counter within the report group just produced is added to each SUM counter in any higher level control footing report group which names the SUM counter just produced. Then, unless reset is suppressed with the RESET clause, each SUM counter at the level just produced is reset to zero. If RESET is specified, the higher level SUM counters are updated, but the SUM counters at the level just produced are not reset to zero.
- The last two steps are repeated as necessary for each level control footing until the level is reached at which the original control break occurred.

Example:

```
01  TYPE IS CONTROL FOOTING SECT-NO  LINE PLUS 2.
    02 COLUMN 20  PICTURE X(10) VALUE IS 'SEC TOTAL'.
    02 COLUMN 30  PICTURE ZZZ,ZZZ.99  SUM AMNT-NO.
```

This clause may be used in an elementary item in a control footing report group to override the automatic resetting of the SUM counter following the associated control break.

$$\underline{\text{RESET}}\ \text{ON}\ \left\{\begin{matrix}\text{data-name-7}\\ \underline{\text{FINAL}}\end{matrix}\right\}$$

It can be used only at an elementary item level in conjunction with the SUM clause. Data-name-7 must be one of the data names described in the CONTROL clause in the RD entry, and it must be a higher level data-name than the CONTROL data-name associated with the control footing report group containing the SUM and RESET clauses.

When RESET is not specified, the SUM counters are automatically reset to zero after the control footing report group is presented. RESET prevents the automatic resetting of the SUM counters until the control footing report group associated with data-name-7 has been presented. This clause permits progressive totaling of data while presenting subtotals at lower levels of control.

RESET FINAL indicates that the counter is not to be reset until the final control footing report group has been printed. With this option, cumulative totals will be presented throughout the report.

Example:

```
RD  REPORT-A
    CONTROLS ARE FINAL, DEPT, SECT, GROUP, MAN.
    .
    .
    .
    01  GROUP-TOTALS TYPE IS CONTROL FOOTING GROUP
        LINE NUMBER IS PLUS 2.
        03  COLUMN 30 PICTURE 9(10)
            SUM GRP-HRS RESET ON SECT.
```

GRP-HRS are summed for this control footing group and the subtotal is incremented and presented with each group total until there is a control break at the SECT level of the control hierarchy. The subtotal is added to the sum at the SECT level and then reset to zero before the SECT control footing report group is presented.

5.5 PROCEDURE DIVISION

To produce a report defined in the Report Section, three Procedure Division statements are required: INITIATE, GENERATE, and TERMINATE. A USE BEFORE REPORTING statement may introduce a Declarative Section through which the user may manipulate, alter, or inspect data immediately before it is printed.

This statement initiates all counters prior to producing a report and begins the processing of a report.

$$\underline{\text{INITIATE}} \left\{ \begin{array}{l} \text{report-name-1 [report-name-2]...} \\ \underline{\text{ALL}} \end{array} \right\}.$$

The report-names are the reports to be initiated. Each name must be defined by a Report Description (RD) entry in the Report Section. ALL specifies that all report-names defined by RD entries in the Report Section of this program are to be initiated.

Only one INITIATE statement can be executed for each report name. If a report name is to be initiated a second time, an intervening TERMINATE statement must be executed for that report name. The INITIATE statement does not open the file with which the report is associated. An OPEN statement for the file must be executed. INITIATE performs Report Writer functions for individually described reports analogous to the input-output functions that OPEN performs for individually described files. These functions are:

- Set all SUM counters to zero; all data name entries containing SUM clauses associated with this report are set to zero.
- Set up the control hierarchy; controls are set up for all report groups associated with this report according to types.
- Preset the page-counter; the page-counter is set to one prior to or during execution of INITIATE. If some other initial value is desired, the user may reset this counter following execution of INITIATE.
- Preset the line-counter; if specified, the line-counter is set to zero prior to or during execution of INITIATE.

If a report heading group contains SOURCE clauses, the values of the specified data-names used in the report heading are the values during execution of the INITIATE statement.

GENERATE

This statement links the Procedure Division to the Report Writer at process time. It presents a report entry under Procedure Division control.

GENERATE data-name-1 .

If data-name-1 names a detail report group, GENERATE handles all relevant automatic operations and produces an output detail report group. This is called detail reporting.

If data-name-1 names an RD entry, GENERATE handles all the relevant automatic operations and updates the footing report groups in the report without producing an actual detail report group (Summary reporting). If the report includes more than one detail report group, all SUM counters are incremented each time GENERATE is executed.

A GENERATE statement, implicitly in both detail and summary reporting, produces the following automatic operations as needed:

- Steps and tests the line-counter and/or page-counter to produce page or overflow footing and/or page or overflow heading report groups.
- Recognizes any specified control breaks to produce control footing and/or control heading report groups.
- Accumulates into the SUM counters all specified data-names. Resets the SUM counters at control break. Performs an updating procedure between control break levels for each set of SUM counters.
- Executes any specified routines defined by USE before generation of the report groups.

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: final, major,and minor, immediately followed by any detail report group specified in the statement. If a control break is recognized when a GENERATE statement is executed (other than the first for a report), all control footing report groups specified for the report are produced from the minor report group up to and including the report group specified for the data-name which caused the control break. The control heading report groups specified for the report, from the report group specified for the data-name which caused the control break down to the minor report group, are then produced in that order. The detail report group specified in the GENERATE statement is then produced.

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

This statement ends processing of a report.

```
TERMINATE { report-name-1 [report-name-2]... } .
          { ALL                               }
```

The report names are reports to be terminated. Each report name must be defined by a Report Description (RD) entry in the Report Section of the Data Division. ALL specifies that all report names defined in the RD entries are to terminate.

TERMINATE produces all the control footing report groups associated with the report as if a control break had just occurred at the highest level (FINAL), and completes the Report Writer functions for the named reports.

TERMINATE produces the appropriate page and report footings for the named reports including the last page footing and report footing associated with each report. Appropriate page/overflow heading and/or footing report groups are prepared in their respective order for the report description.

If SOURCE clauses are included in the final control footing or report footing groups, the values for the SOURCE data-names are the values of the data items during execution of the TERMINATE statement.

A second TERMINATE for a particular report may not be executed unless an intervening INITIATE has been executed for that report. TERMINATE does not close the file with which the report is associated; the CLOSE statement for the file must be specified by the user.

USE BEFORE REPORTING

This statement must follow a section header in the Declarative Section. It introduces procedures to be performed immediately before the specified report groups are produced.

```
USE BEFORE REPORTING identifier-1 [identifier-2]...
```

The identifiers may be any type report group (01 level) except DETAIL. An identifier must not appear in more than one USE statement. The report writer verbs, GENERATE, INITIATE, TERMINATE, may not be used in any procedure introduced by USE BEFORE REPORTING.

This USE implies a PERFORM of the sentences between the USE statement and the end of the section or the END DECLARATIVES for each report group specified by the identifiers. The sentences are executed immediately before the report group is produced after any summing, and after data has been moved into the line image, but before the line-counter is incremented. All logical paths within this section of the declaratives must lead to a common exit point. PERFORM, GO TO or ALTER are the only statements that may reference procedure names within the Declarative Section, in another Declarative Section, or in the non-declarative part of the program.

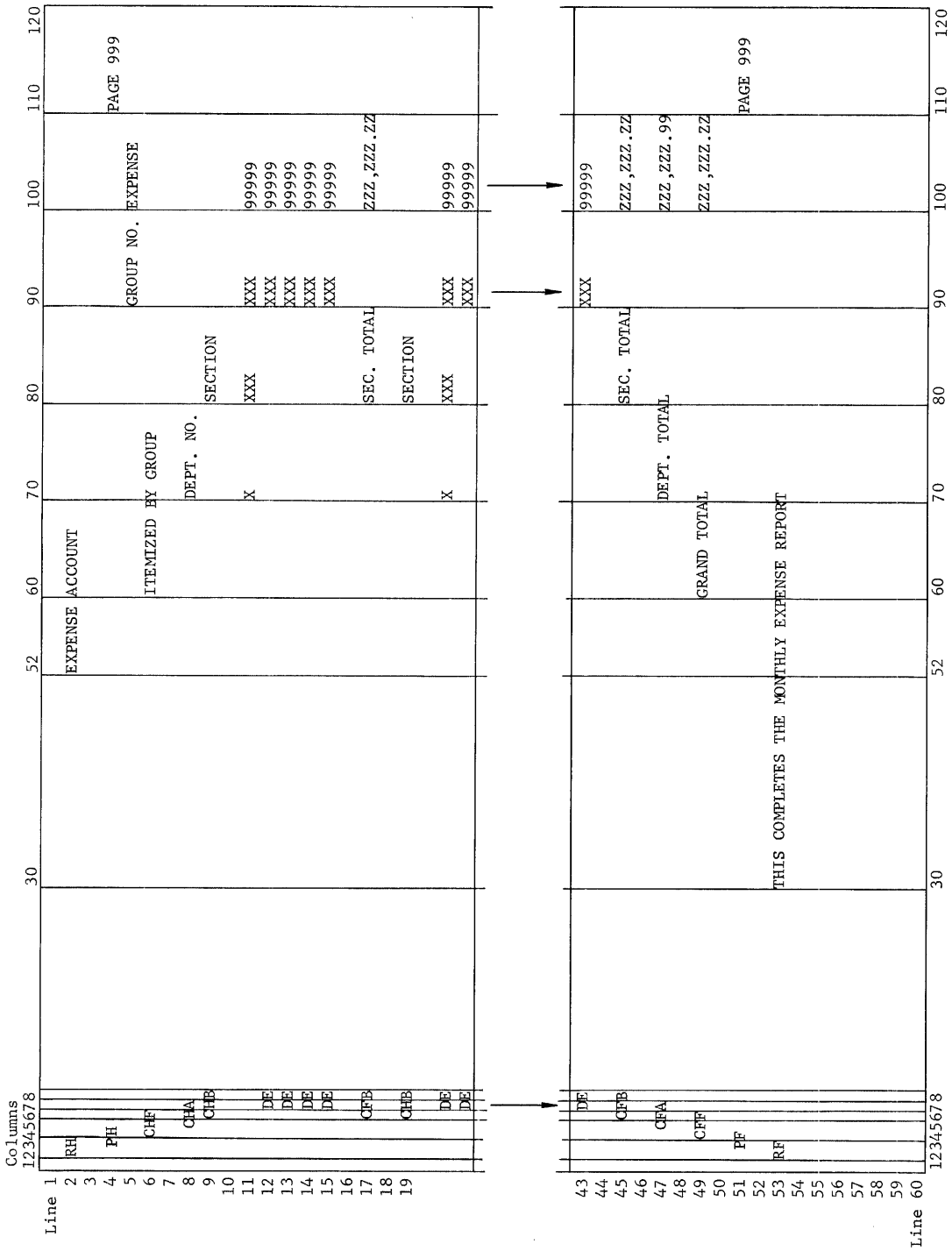
USE BEFORE REPORTING is particularly helpful to check whether summing is correct, or to add to or alter the SUM counter.

5.6 SAMPLE REPORT WRITER PROGRAM

```
IDENTIFICATION DIVISION.
PROGRAM-ID. REPORT-WRITER-EXAMPLE.
AUTHOR. CDC.
DATE-WRITTEN. MARCH 7, 1967.
DATE-COMPILED.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. 6400.
OBJECT-COMPUTER. 6400.
INPUT-OUTPUT SECTION.
FILE-CONTROL. SELECT FILE-B ASSIGN TO OUTPUT.
    SELECT FILE-A ASSIGN TO TAPE-01.
DATA DIVISION.
FILE SECTION.
FD FILE-A
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS CARD-IMAGE.
    01 CARD-IMAGE.
       02 DEPT-NO PICTURE X(5).
       02 SECT-NO PICTURE X(5).
       02 GROUP-NO PICTURE X(5).
       02 AMNT-NO PICTURE 9(5).
FD FILE-B
    LABEL RECORDS ARE OMITTED
    REPORT IS REPORT-A.
REPORT SECTION.
RD REPORT-A
    CONTROLS ARE FINAL, DEPT-NO, SECT-NO
    PAGE LIMIT IS 60 LINES
    HEADING 1
    FIRST DETAIL 10
    LAST DETAIL 55
    FOOTING 60.
    01 TYPE IS REPORT HEADING LINE 02.
       02 COLUMN 02 SIZE 2 VALUE IS 'RH'.
       02 COLUMN 52 SIZE 15 VALUE IS 'EXPENSE ACCOUNT'.
    01 TYPE IS PAGE HEADING.
       02 LINE PLUS 02.
          03 COLUMN 03 SIZE 2 VALUE IS 'PH'.
          03 COLUMN 110 SIZE 4 VALUE IS 'PAGE'.
          03 COLUMN 115 PICTURE 999 SOURCE IS PAGE-COUNTER.
       02 LINE PLUS 01
          03 COLUMN 90 SIZE IS 8 VALUE IS 'GROUP NO'.
          03 COLUMN 100 SIZE IS 7 VALUE IS 'EXPENSE'.
    01 TYPE IS OVERFLOW HEADING LINE PLUS 02.
       02 COLUMN 03 SIZE IS 2 VALUE IS 'OH'.
       02 COLUMN 20 SIZE IS 28 VALUE IS 'CONTINUED FROM PREVIOUS PAGE'.
       02 COLUMN 110 SIZE IS 4 VALUE IS 'PAGE'.
       02 COLUMN 115 PICTURE 999 SOURCE IS PAGE-COUNTER.
    01 TYPE IS CONTROL HEADING FINAL LINE PLUS 02.
       02 COLUMN 04 SIZE 03 VALUE IS 'CHF'.
       02 COLUMN 60 SIZE IS 17 VALUE IS 'ITEMIZED BY GROUP'.
    01 TYPE IS CONTROL HEADING DEPT-NO LINE PLUS 02.
       02 COLUMN 05 SIZE 03 VALUE IS 'CHA'.
       02 COLUMN 70 SIZE IS 7 VALUE IS 'DEPT-NO'.
```

```
       01 TYPE IS CONTROL HEADING SECT-NO LINE PLUS 01 NEXT GROUP PLUS 2.
          02 COLUMN 06 SIZE IS 03 VALUE IS 'CHB'.
          02 COLUMN 80 SIZE IS 7 VALUE IS 'SECTION'.
       01 DETAIL-ITEM TYPE DETAIL LINE PLUS 1.
          02 COLUMN 07 SIZE IS 2 VALUE IS 'DE'.
          02 DEPT PICTURE X COLUMN 70 SOURCE DEPT-NO
             GROUP INDICATE.
          02 SECT PICTURE X(3) COLUMN 80 SOURCE SECT-NO
             GROUP INDICATE.
          02 GRP-N PICTURE X(3) COLUMN 90 SOURCE GROUP-NO.
          02 AMOUNT PICTURE 9(5) COLUMN 100 SOURCE AMNT-NO.
       01 TYPE IS CONTROL FOOTING SECT-NO LINE PLUS 02.
          02 COLUMN 06 SIZE IS 03 VALUE IS 'CFB'.
          02 COLUMN 80 PICTURE X(10) VALUE IS 'SEC TOTAL'.
          02 COLUMN 100 PICTURE ZZZ,ZZZ.ZZ SUM AMNT-NO.
       01 TYPE IS CONTROL FOOTING DEPT-NO LINE PLUS 02.
          02 COLUMN 05 SIZE IS 03 VALUE IS 'CFA'.
          02 COLUMN 70 PICTURE X(10) VALUE IS 'DEPT. TOTAL'.
          02 COLUMN 100 PICTURE ZZZ,ZZZ.99 SUM AMNT-NO.
       01 TYPE IS CONTROL FOOTING FINAL LINE PLUS 02.
          02 COLUMN 04 SIZE 03 VALUE IS 'CFF'.
          02 COLUMN 60 PICTURE X(11) VALUE IS 'GRAND TOTAL'.
          02 COLUMN 100 PICTURE ZZZ,ZZZ.ZZ SUM AMNT-NO.
       01 TYPE IS OVERFLOW FOOTING LINE PLUS 02.
          02 COLUMN 03 SIZE IS 02 VALUE IS 'OV'.
          02 COLUMN 20 SIZE IS 22 VALUE IS 'CONTINUED ON NEXT PAGE'.
       01 TYPE IS PAGE FOOTING LINE PLUS 02.
          02 COLUMN 03 SIZE IS 02 VALUE IS 'PF'.
          02 COLUMN 110 SIZE IS 4 VALUE IS 'PAGE'.
          02 COLUMN 115 PICTURE 999  SOURCE IS PAGE-COUNTER.
       01 TYPE IS REPORT FOOTING LINE PLUS 02.
          02 COLUMN 02 SIZE IS 02 VALUE IS 'RF'.
          02 COLUMN 30 SIZE IS 41 VALUE IS 'THIS COMPLETES THE
                                           'MONTHLY EXPENSE REPORT'.
PROCEDURE DIVISION.
START.
    OPEN INPUT FILE-A.
    OPEN OUTPUT FILE-B.
    INITIATE REPORT-A.
STEP-2.
    READ FILE-A AT END GO TO STOP-IT.
    GENERATE DETAIL-ITEM.
    GO TO STEP-2.
STOP-IT.
    TERMINATE REPORT-A.
    CLOSE FILE-A, FILE-B.
    STOP RUN.
```

The data for the report produced by the sample program is input on FILE-A in card format, and is output on FILE-B. The printed format of the report at the highest (FINAL) control break is shown below:


Columns
12345678
30
52
60
70
80
90
100
110
120
Line 1
Line 60
RH
EXPENSE ACCOUNT
PH
ITEMIZED BY GROUP
CHF
DEPT. NO.
SECTION
GROUP NO. EXPENSE
PAGE 999
CHA
CHB
X
XXX
99999
DE
ZZZ,ZZZ.ZZ
CFB
SEC. TOTAL
CHB
DE
CFA
DEPT. TOTAL
CFF
GRAND TOTAL
ZZZ,ZZZ.99
PF
THIS COMPLETES THE MONTHLY EXPENSE REPORT
RF

SCOPE/COBOL INTERRELATION 6

6400/6500/6600 COBOL runs under control of the SCOPE system. The COBOL programmer should be familiar with four general areas of SCOPE control:

Input/output control; including file and label handling

Compilation and subcompilation of source programs

Execution of object programs

Library preparation and maintenance

6.1 INPUT-OUTPUT CONTROL

6.1.1 SCOPE FILES

File Name

All files immediately available to the SCOPE system at any time are called active files and are listed in the File Name/File Status table resident in central memory. A logical file name of one to seven display code characters is associated with each active file. Whenever the COBOL user specifies a SCOPE file as an implementor name in the SPECIAL-NAMES paragraph or the FILE-CONTROL paragraph of the Environment Division, this name must be 1-7 characters in length. Each SCOPE file name used in a COBOL source program must be unique. A maximum of 53 files are allowed the COBOL user, including SCOPE system files, FD entries, SD entries, and report file RD entries.

Special System Files

The special SCOPE system files INPUT, OUTPUT, PUNCH, and PUNCHB are available to the COBOL user. They may be specified wherever the user may specify a SCOPE file as an implementor name. INPUT is the immediate source of card input, OUTPUT is the immediate destination of printer output, and PUNCH (Hollerith) and PUNCHB (binary) are the immediate destination of card output. When a job terminates, INPUT is released to the system and the three output files are assigned to the job just terminated.

File Format

All SCOPE files are organized into logical records. A SCOPE logical record is always equal to one or more physical records. The physical record size of SCOPE files is determined by the particular device. The BLOCK CONTAINS specifies size in terms of characters, this size should be at least as large as the minimum physical record size for the particular hardware device. If the clause is omitted, the system assigns 512 words to a block.

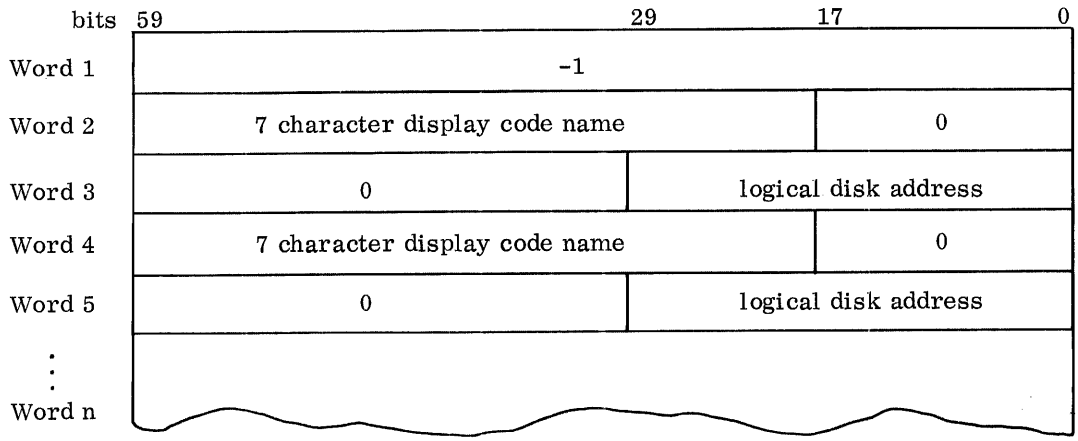
The size required by the hardware devices is shown below:

Minimum Size	Hardware Device
64 words	6638 Disk
64 words	6603 Disk
64 words	Card Punch
64 words	Line Printer
128 words	1/2-inch Magnetic Tape (Display Code/BCD)†
512 words	1/2-inch Magnetic Tape (Binary)
512 words	1-inch Magnetic Tape (Binary)

6.1.2 FILE INDEX

Every file specified as ACCESS MODE IS RANDOM has a file index. If the SYMBOLIC KEY clause is specified, this index contains the names and addresses of the records in the file. If the ACTUAL KEY clause is specified, it contains only the addresses of the records. The name and number indexes are shown below:

Name Index:

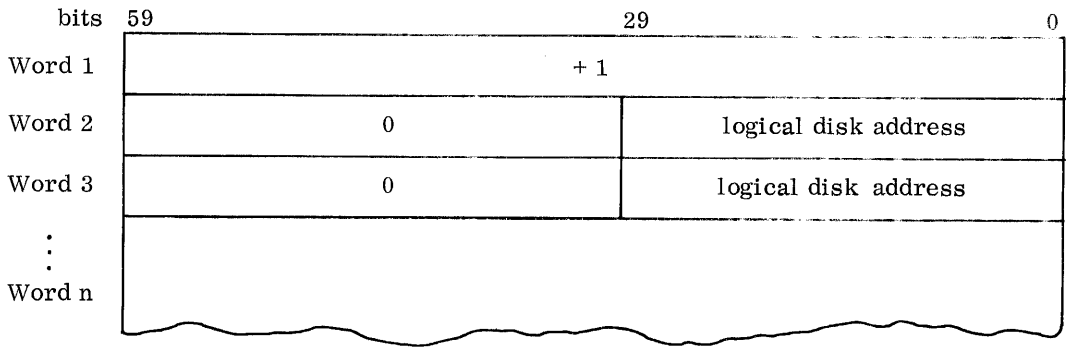


Each random access record with a symbolic key has one two-word entry in the name index. Word one of the name index is set to -1 when the file is opened; it is zero until the file is opened.

† Within the SCOPE system, all coded information is carried in display code and converted to external BCD before writing on coded tape. (See Appendix B for the Conversion).

Each record name is 1-7 characters left justified with zero fill in bits 0-17 of the first word of the entry. Its corresponding logical disk address is placed in bits 0-29 of the second word of each entry; it is a pointer to system tables where the actual address is found. When a new record is written with the same name as the one that is already in the index, the new record takes the place of the existing record. If the name is not already in the index, the system puts the name in the lowest numbered position with no name, and assigns the number of that position to the record. If there is no room for a new name, the index is full and an INVALID KEY will be executed. When a record is read by name, the name must be present in the index or an INVALID KEY is executed.

Number Index:



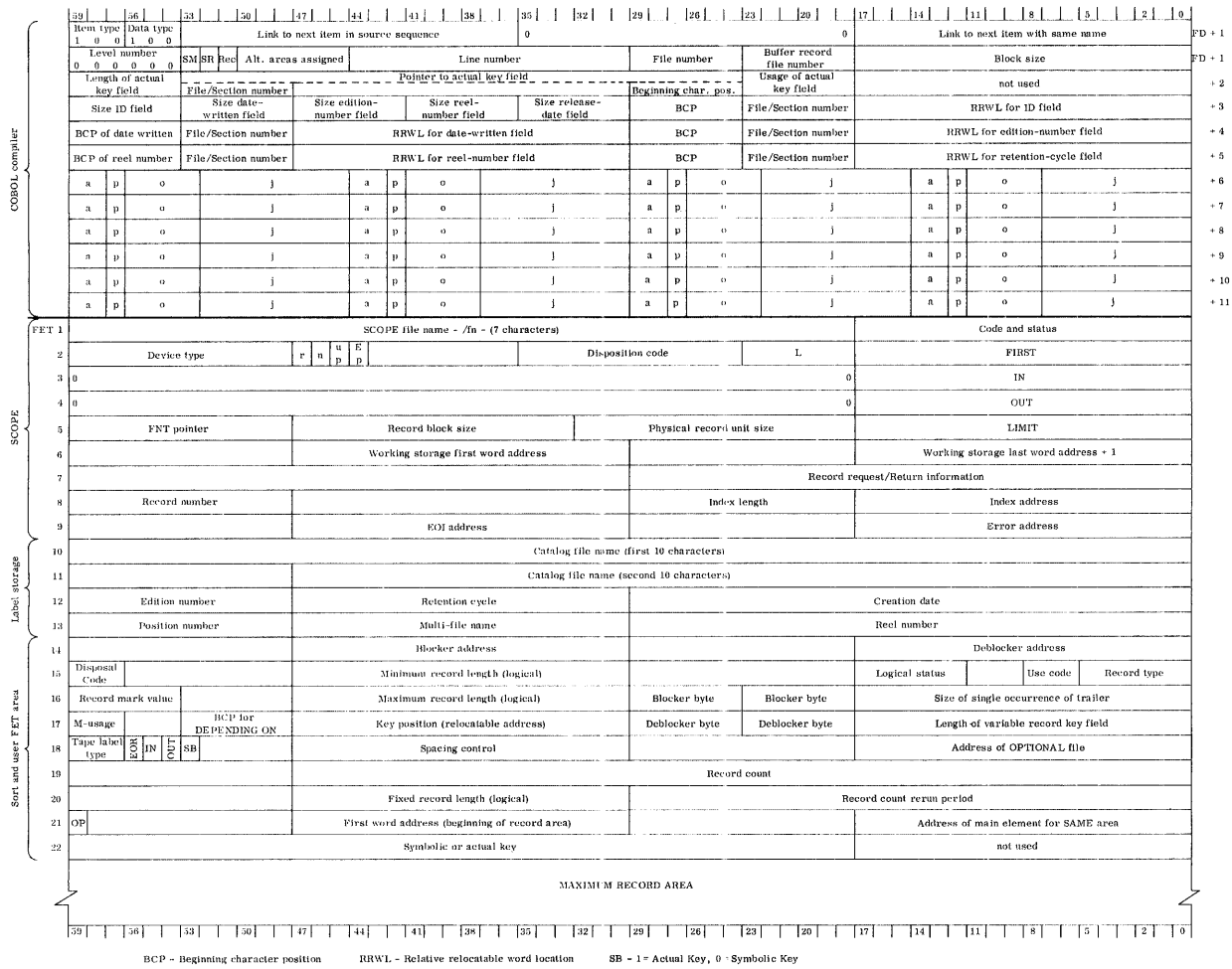
Each random access record with an actual key has a one-word entry in the number index. Word 1 of the number index is set to +1 when the file is opened; it is zero until the file is opened. In the index the logical disk address of a record is in bits 0-29 of the word assigned to the record. For an actual key record, the key must be computed by the user to result in a positive integer which is the index position in the number index for the record.

6.1.3 FILE ENVIRONMENT TABLE

Each file index, whether a number or name index, has corresponding entries in the File Environment Table (FET) for the file. The FET is generated by SCOPE for every file name specified in a COBOL source program. In addition to file index information, it also contains label information for tape files with standard labels: the file name, device type, physical record unit size, and so forth. The format of the FET is shown on page 6-4 .

The augmented FET table has three main parts: The COBOL compiler uses the words FD+1 through FD+11. The SCOPE system uses the first 13 words of the actual FET, FET 1 through FET 13; and the SORT system and the COBOL user use words FET 14 through FET 20.

The COBOL compiler sets the fields in FD+1 through FD+13 from information supplied by the source program. This section is self explanatory except for words FD+6 through FD+11. These words



BCP - Beginning character position RRWL - Relative relocatable word location SB - 1 = Actual Key, 0 = Symbolic Key

describe any USE procedures to the COBOL I/O package. Each word contains four entries consisting of the following items:

a = 0 Use with File
1 INPUT
3 OUTPUT
4 INPUT and OUTPUT

p = 0 USE BEFORE
1 USE AFTER

o = 00 Error Procedure
05 Ending File Label
06 Ending Reel Label
07 Ending File/Reel Label
11 Beginning File Label
12 Beginning Reel Label
13 Beginning File/Reel Label
15 Beginning and Ending File Label
16 Beginning and Ending Reel Label
17 Beginning and Ending File/Reel Label

j = Location within Jump Table of the Base Package

In the part of FET used by SCOPE (FET 1 through FET 13), the file indexing fields, FET 7 and 8, and the fields used with labeled tape files, FET 10 through 13, are of most concern to the COBOL user.

During random file processing, the record request/return information field is set to the location of the word in the file index containing the logical disk address of the record being processed. The record number field contains a binary integer, indicating the last logical record read or written. The index length field contains a positive integer. It is used by SCOPE to define the amount of memory saved on disk for the index when a CLOSE OUTPUT is executed, and to bring the index back to memory from disk when an OPEN OUTPUT is executed. The index address field is set to the location of the file index in memory. This location is an absolute address relative to the relocatable address (RA) of the beginning of memory.

FET 10 through 13 are set to the contents of a standard label specified by the user in the VALUE OF clause of the LABEL RECORDS. These words are set only for standard SCOPE labels. If the UP bit (FET 2) is zero, all label processing proceeds automatically; if the UP bit is set, USE routines have been specified to perform additional checking in conjunction with standard SCOPE label routines.

FET 14 through 22 are set by SORT and the COBOL user. They contain file control and sort file information. Generally, these entries are self explanatory. A detailed description of all the FET fields are in the 6400/6500/6600 SCOPE Reference Manual.

6.1.4 STANDARD LABELS

SCOPE labels are specified as STANDARD in the LABEL RECORDS clause of an FD description. Standard labels are provided by SCOPE only for files written on 1/2-inch magnetic tape. The labels are in the coded mode. The user must request such a file with a REQUEST card (See Equipment Assignment). Each label is checked by the system for identification by file name, reel number, creation date, expiration date, and edition number. SCOPE does not provide label processing for non-standard system labels.

Labels are of four types:

Volume Header Label	(VOL1)
File Header Label	(HDR1)
Volume Trailer Label	(EOV1)
File Trailer Label	(EOF1)

In this context, volume is a reel of magnetic tape; thus, the volume header label is equivalent to a beginning reel label. One volume may contain one file or several files; several consecutive volumes may contain one file or several files.

Each type of label is separated from the data in a file by a tape mark, which consists of a one character record (17_8 external BCD) plus a check character recorded in even parity.

Every reel of tape is preceded by a volume header label which is the first physical record in the volume. Every file is preceded by a file header label. If a volume ends within a file (multi-reel-file), the continuation of the file into the next volume is also preceded by a file header label following the volume header label. Every file header label is immediately followed by a tape mark (*). The last record of every file is followed by a file trailer label preceded and followed by a tape mark. The last file trailer label in a volume is followed by two tape marks. A volume trailer label is used whenever a volume ends within a file (multi-reel file). The last physical record of the file in a volume is followed by a volume trailer label preceded by one tape mark and followed by two tape marks.

The following shows possible organization of labels, data, and tape marks on a magnetic tape file:

Single-Reel File

VOL1 HDR1 * . . . Data Blocks . . . * EOF1 * *

Multi-File Reel

VOL1 HDR1 * . . . File A . . . * EOF1 * HDR1 * . . . File B . . . * . . . EOF1 * *

Multi-Reel File

VOL1 HDR1 * . . . First Volume Data . . . * EOV1 * *

VOL1 HDR1 * . . . Last Volume Data . . . * EOF1 * *

Multi-Reel Multi-File

VOL1 HDR1 * . . . File A . . . * EOF1 * HDR1 * . . . File B . . . * EOV1 * *

VOL1 HDR1 * . . . Continuation of File B . . . * EOV1 * *

VOL1 HDR1 * . . . Last of File B . . . * EOF1 * HDR1 * . . . File C . . . * EOF1 * *

Whenever the end of a reel and the end of a file coincide, the label configuration is one of the following:

. . . File A . . . * EOV1 * *

VOL1 HDR1 * * EOF1 * HDR1 * . . . File B . . .
(A) (A) (B)

. . . File A . . . * EOF1 * HDR1 * * EOV1 * *
(A) (B)

VOL1 HDR1 * . . . File B . . .
(B)

Label Record Formats

A standard label is one 80-character physical record. In the formats below:

n = any numeric digit 0-9

a = any character in the COBOL character set in DISPLAY format.

Volume Header Label

Character	Content	Description
1-3	VOL	Label identifier; volume header
4	1	Label number
5-10	nnnnnn	Visual reel number; 6 characters giving the number stamped on the tape reel.
11	a	Security: blank, not protected. non blank, protected.
12	a	Volume density; blank or 0 = 556 BPI 1 = 200 BPI 2 = 800 BPI
13-80	spaces	Reserved for system use

File Header Label

Character	Content	Description
1-3	HDR	Label identifier; file header
4	1	Label number
5-24	20(a)	File label name; up to 20-character identification, it must begin with a letter.
25-27	aaa	File set ident.; up to three characters starting with a letter. The same identification must be used for every file in the multi-file set.
28-31	nnnn	Reel number; 0001 to 9999, incremented by 1 after each volume trailer label is written.
32-35	nnnn	Multi-file position number; gives position of file in a set of files.
36-39	spaces	Reserved for system use
40-41	nn	Edition number; 00 to 99, distinguishes different editions of same file.
42	space	Reserved for system use
43-47	YYDDD	Creation date; YY = year, DDD = Julian Date (001 to 366)
48	space	Reserved for system use
49-53	YYDDD	Expiration date. When this is equal to or less than today's date, the file is expired and the volume may be erased.
54	a	Security; blank = not protected non-blank = protected
55-60	zeros	Block count
61-80	spaces	Reserved for system use

File Trailer Label

Character	Content	Description
1-3	EOF	Label identifier; end of file
4	1	Label number
5-54	optional	Identical to corresponding characters in file header label.
55-60	nnnnn	Block count; number of physical records including labels and tape marks since last file header label.
61-80	spaces	reserved for system use

Volume Trailer Label

Character	Content	Description
1-3	EOV	Label identifier; end of volume
4	1	Label number
5-54	optional	Identical to corresponding characters in volume header label.
55-60	nnnnn	Block count, number of physical records excluding labels and tape marks since last volume header label.
61-80	spaces	Reserved for system use

6.1.5 NON-STANDARD LABELS

SCOPE standard labels are provided and checked on 1/2-inch magnetic tape. The user may prepare and check his own 1/2-inch magnetic tape labels (non-standard) by specifying a data-name in the LABEL RECORDS clause of the File Description Entry (FD). This data name identifies the label which the user may write on an output file or read from an input file.

If this option is used, the first physical record of the file is assumed to be the record defined by the data-name entry. It is read or written when the OPEN statement is executed prior to the first READ or WRITE for the file. It can be referenced by data-name in the same way as any other data record in the program. OPEN INPUT corresponds to OPEN INPUT with READ INTO data-name-2, and OPEN OUTPUT corresponds to OPEN OUTPUT with WRITE FROM data-name-2. USE procedures may be included for label checking with non-standard labels, but they are not necessary.

End-of-reel labels on input files may be checked automatically only if the VALUE OF ENDING-TAPE-LABEL-IDENTIFIER clause is specified in the LABEL RECORDS clause. This identifier is used to distinguish between end-of-file and end-of-reel labels. If this option is used, USE procedures referring to reels may be applied for input tapes. They are executed under the following conditions:

Before Ending Reel

This procedure is executed after reading the tape mark and prior to reading the label record when it is not yet determined, whether it is end-of-reel or end-of-file. Therefore, this is not a very useful procedure.

After Ending Reel And Before Beginning Reel

These procedures are executed after the first 40 characters of the label record have been read into the TAPE-LABEL area. (This area may be referenced at any time from the Procedure Division, which corresponds to words 10-13 of the File Environment Table).

At this point, it has not yet been determined whether the tape mark is end-of-reel or end-of-file. These procedures could be used to make a test of the first 40 characters of the ending label and to change them if desired.

After Beginning Reel

This procedure is executed after reels have been swapped. The first 40 characters of the first record (label) of the new reel are in the TAPE-LABEL area (execept for the first reel of a file).

There is no built-in facility to handle non-standard label multi-reel files for output tapes. Sensing of physical end-of-reel on a file being written without standard labels will force the file to be closed. Any applicable USE BEFORE ENDING FILE LABEL PROCEDURE will be executed first, however. If such a procedure does not exist or does not close the file, it will be automatically closed with rewind.

A programmer can employ a USE BEFORE/AFTER ENDING FILE procedure if he wishes to be prepared for the end of tape. He will be informed of reaching the end of tape by execution of the USE procedure, and he will then have the opportunity to execute a CLOSE or CLOSE WITH NO REWIND for the file. Otherwise, the compiler will automatically execute a close, rewinding the tape.

6.1.6 OMITTED LABELS

If labels are specified as OMITTED in the LABEL RECORDS clause, it is assumed that the user is not concerned with labels or that he intends to handle the label by means of source program coding. The OPEN, READ, and WRITE requests function in the normal manner. Following the first OPEN, a label record will not be available in the label record area. The first READ transfers the first logical record directly to the data record area, and the first WRITE writes the logical record in the record area onto the file. If this first record is to be treated as a label, it must be handled exactly as any other logical record in the file.

6.2 COMPILATION

The COBOL source program is written on a COBOL coding sheet according to the specifications in this manual. (Coding rules are summarized in Appendix A.) Information on the coding sheets is punched character for character on standard 80-column cards, one line for each card. The resulting source program card deck is used as input to the COBOL compiler. It is presented in the following order: IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, DATA DIVISION, and PROCEDURE DIVISION. Depending on the requirements of the user, additional control cards are included with the source program deck to control the form of compilation.

Compilation may be accomplished without a resulting object program, or it may be followed immediately by execution of the object program. In addition, sections of a program may be compiled separately and then linked by the SCOPE loader into one object program. This subcompile capability allows a single large COBOL program to be created at object time from several smaller COBOL programs.

6.2.1 OUTPUTS FROM COMPILATION

The outputs from compilation include a listing of the source program with error diagnostics, listing of the object programs, and the binary object program itself ready for execution. The user may

indicate no object program output is required, the object program is to be written on a specified file, or object program cards are to be produced for later execution.

Source Program Listing

Each line on a printed listing of the source program corresponds to one card in the source program deck or to one line on the COBOL coding sheet. The format and order of each line on the listing is identical to that of the COBOL coding sheet. Within this listing, errors in the source program are indicated by a diagnostic message and a code number. Diagonstics are preceded by a series of asterisks. The code number is associated with an error condition. Appendix G includes the codes, error conditions, and corrective measures.

This listing with major diagnostics is the normal listing produced with every compilation. It may be suppressed or additional output may be requested with the list parameter of the COBOL control card.

Additional Outputs from Compilation

Cross reference pointers to source lines

Listing of items copied from library

Object program in relocatable binary form writted on specified file

Octal listing of object program

Listing of extended diagnostics

Suppression of all listing

Data map

If a list of items copied from the library is requested, they will appear in the listing at the point where they occur; following either a COPY clause or an INCLUDE statement.

If the object program is requested on a load and go file, it is ready for execution with real data or test data. Any correction to the program should be made at the source program level.

An octal listing of the object program can be obtained but it is of no particular value to the user since all corrections should be made at the source program level. The object program listing is primarily an aid to the maintenance of the system and to debugging.

The extended diagnostics listing includes messages with code letters T and U; they do not drastically affect the running of the program. These trivial diagnostics usually point out errors in syntax or moves that require additional time or space. They will not cause program termination but may be useful in improving the execution time of a program. No diagnostics recognize errors in programmer logic unless the error also results in an infraction of one or more rules of the COBOL language.

Suppression of all listing may be requested. This is primarily useful when the program is out of the debugging stage and no changes are expected in the COBOL source program.

When a data map is requested, it will appear between the source listing of the Data Division and the source listing of the Procedure Division. This map is essentially a map of the Data Division and covers fields in the File Section, the Common-storage Section, the Working-storage Section, and the Constant Section. Line number, level number, name, and location relative to the beginning of the file or section are given for each field.

6.2.2 COBOL CONTROL CARD

The COBOL control card calls the COBOL compiler, specifies the files to be used for input and output, and indicates the kind of output to be produced.

The control card consists of the word COBOL optionally followed by a parameter list enclosed in parentheses. The card columns following the right parentheses may be used for comments, they are ignored by the system.

If no parameter list is given, a period must be used to separate the word COBOL from any comments on the card. If no comments are given, the period is unnecessary. Blank columns may be used anywhere for readability; they will be squeezed out and ignored by the system. The formats for the control card are given below.

COBOL (p_1, p_2, p_3, p_4, p_5, p_6) comments

COBOL. comments

The parameters may be any or all of the following:

p_1 Source Input Parameter

If no source input parameter is present, file INPUT is assumed. If the source input is a file other than INPUT, a source input parameter of the following form must be provided:

I = fn

fn is the file name where the source input appears.
I = INPUT, and I, are equivalent to omitting this parameter.

p_2 Binary Output Parameter

If no binary output parameter is present, a relocatable binary file will be written on a file named LGO. If binary output is on another file, a binary output parameter of the following form must be provided:

B = fn

fn is the file name on which the binary output is to be written. B = 0 will suppress binary output. B = LGO, or B is equivalent to omitting a binary output parameter.

p_3 List Parameter

If no list parameter is provided a normal listing is provided on OUTPUT. It includes source language (excluding library copies) and major diagnostics. Other list options may be selected with the following parameters:

ℓ = fn; ℓ may be one of the following:

L normal listing

L may be suffixed by any combination of the following parameters, to provide the following features in addition to the normal listing.

X extended diagnostics

R cross reference pointers to source lines

C items copied from library

O object code in octal

M data map

fn = file name on which list output is to be written. fn = 0 suppresses all list output. If = fn is omitted, the list output will be on file OUTPUT.

p_4 Source Library Parameter

If no COBOL source library parameter is present and library is required by a COPY or INCLUDE clause in the COBOL source program, the COBOL system will obtain source library information from the file named COLIB. If the COBOL source library is on a file other than COLIB, a COBOL library parameter of the following form must be provided:

S = fn

fn is the file name where the COBOL source library appears.
S = COLIB, and S, are equivalent to omitting the source library parameter.

p_5 Subcompile Parameter

A parameter may be included to suppress certain output for compilations to be used as subprograms with another compilation.

SUB

This parameter suppresses all data division binary output except from working storage and constant storage for a subcompiled program which would duplicate output from a separately compiled main program. This enables them to be loaded together properly.

p_6 Overlay Binary Parameter

This parameter may be included to separate overlay segments from main programs so that separately compiled programs can be loaded properly.

OB = fn

This parameter causes the binary output segments to be put on a file fn. The form OB is equivalent to OB = LGO2.

Examples:

COBOL (LX)

This control card calls for compilation with normal listing plus extended diagnostics on OUTPUT, relocatable binary on LGO, and source input from INPUT. If the source program requests items from the library, they will be taken from the file COLIB.

COBOL (I= MYTAPE, S= SRCLIB, B= 0, LO)

This control card calls for compilation with no binary output, normal listing plus octal object code listing on OUTPUT, and source input from the file named MYTAPE, with SRCLIB as the COBOL source library.

The end of a COBOL source program is indicated by an end of record card (7,8,9 punch in column 1.)

6.2.3 SUBCOMPILE CAPABILITY

When a COBOL system is being prepared, the subcompile feature allows changes or modifications to be made to one program without affecting other programs compiled separately. Each program consists of at least the Identification Division, a File Section, or a Common-Storage Section of the Data Division and the particular procedures in the Procedure Division which are to be compiled separately. The main COBOL program contains the full Data Division and the Environment Division, if required.

Program Identification

The program name in the Identification Division of a program compiled separately must be distinguished from the program name of any other program within the first six characters of the name. The first character must be alphabetic. The nest five may be alphabetic c numeric; a hyphen may not be included. COBOL permits the program name to be up to 30 characters, alphabetic or numeric, but SCOPE places this further restriction on the program name, when the subcompile capability is used.

Entry Points

If a procedure within one program references a procedure within a separately compiled program, entry points must be provided in the referenced program. When a sequence control statement (PERFORM, GO TO, or ALTER) references a procedure name that is not defined in the program containing these statements, SCOPE generates linkage instructions to entry points in a separately compiled program. Entry points are defined by the statement:

ENTRY procedure-name-1 [procedure-name-2]. . .

The procedure names may be up to 30 alphanumeric characters, but each name must be unique within the first six characters and the first must be alphabetic. ENTRY must be the first Declarative in the Declaratives Section of the Procedure Division of the referenced program.

Common Storage and File Areas

When several programs are compiled separately they may communicate through common storage and file areas. Each program must contain its own Common-Storage Section and File Description entries, but only the common storage area and the file and record areas associated with the main program are loaded. In addition, if the SAME RECORD AREA is specified for any files used in common, this clause either must be in all the programs using the file, or it must be in the program loaded first. When USE statements are associated with a file all must be contained in the main program.

Overlay Segments

Priority sections which generate separate overlays segments may appear in any compilations run together (main or subcompiled), but each overlay segment must be wholly contained in one subcompilation.

Overlay Loading

To load overlays, an overlay card must be the first input to the loader. Since the order of compilation is not restricted, the compiler cannot supply this card in the general case. The compiler supplies an overlay card only when neither a subcompile nor overlay binary parameter is present, and then only if overlays are called for by section priority numbers over 50. If SUB or OB is specified on the COBOL card, the programmer may supply the card by copying the following card onto his binary output prior to all compilations:

OVERLAY (COBCODE, 0, 0)

Subcompilation Compatibility

To produce a main program and subcompiled programs that are compatible and can run together without special manipulations, the following restrictions hold:

Restrictions on Main Program

It must contain:

all file descriptions

all report descriptions

all common storage

all USE sections (in DECLARATIVE sections)

a Procedure Division where the program starts

It may contain:

overlay sections (priority numbers must be unique to main program)

working storage (not shared)

constant storage (not shared)

Restrictions on a Subprogram

It must contain:

a complete file description for any file referenced in any way

a complete file description for any file sharing storage with a file whose description is given

a complete report description for any report referenced in any way

a common storage description complete from start to any fields referenced

at lease one ENTRY statement

It may not contain:

any USE sections

It may contain:

overlays (Priority numbers must be unique to this sub program)

working storage (not shared)

constant storage (not shared)

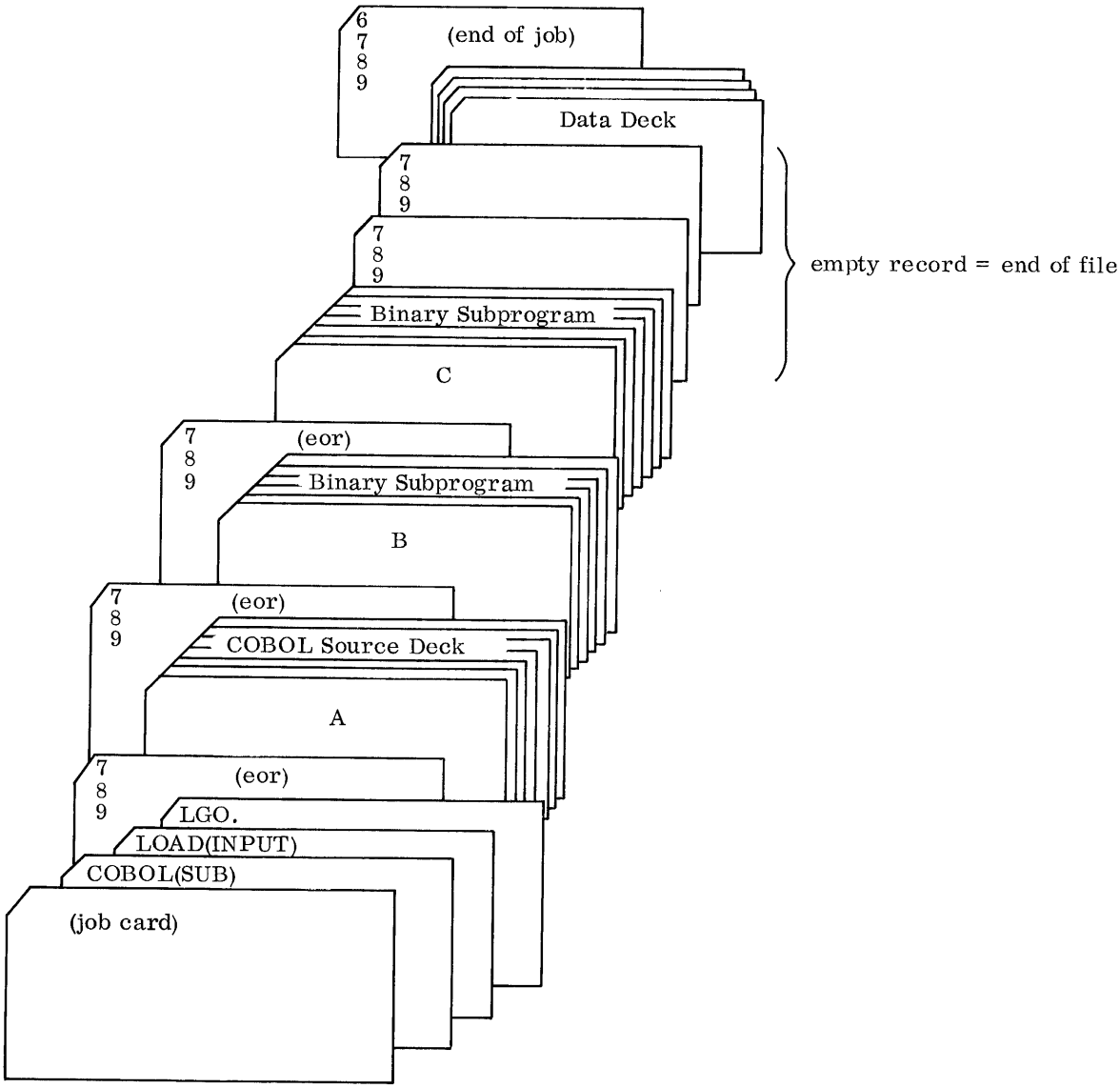
procedures not in overlays

Execution with Subcompile Capability

The order of loading is determined by the execution control cards. The last program call card or

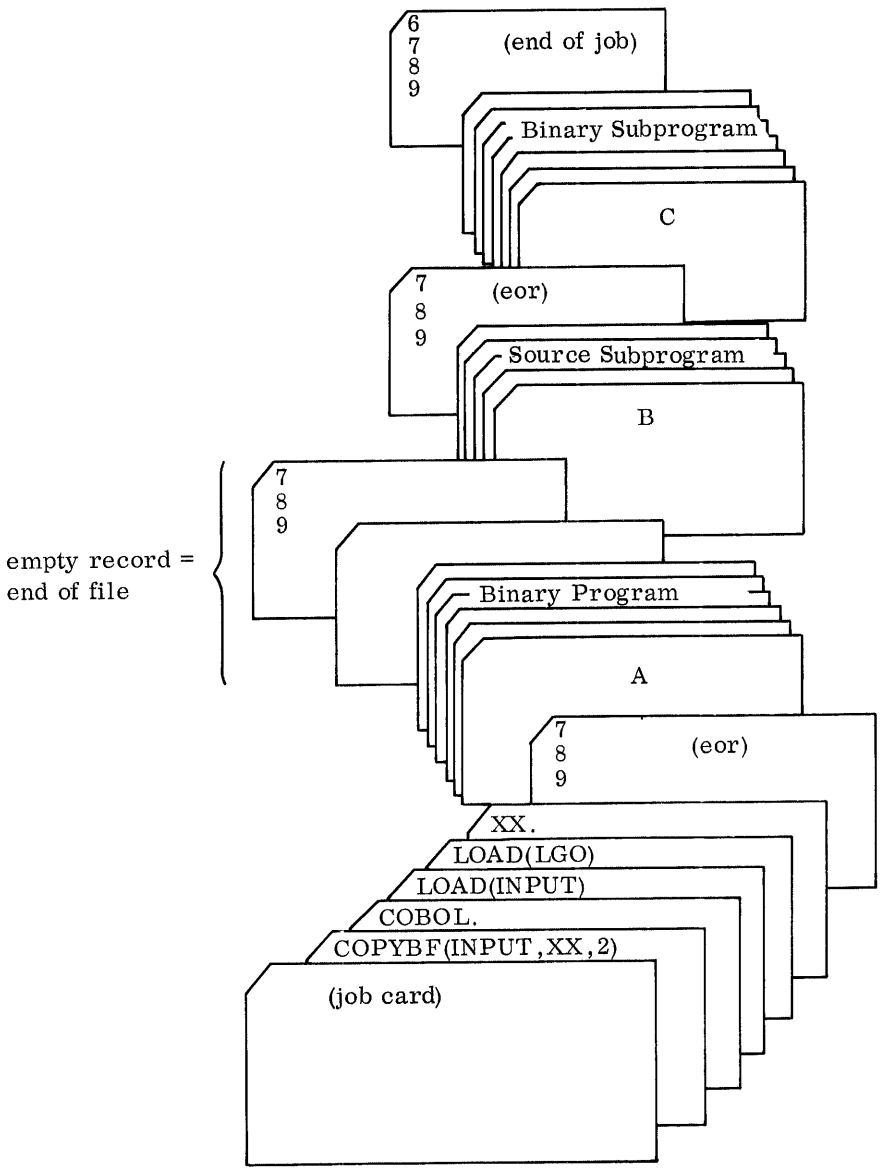
load card specifies the first program loaded, the preceding card specifies the next program loaded, and so forth with the first card specifying the last program loaded. Any EXECUTE card in the deck overrides any other control cards and specifies the first program loaded and its entry points. The execution control cards are described in Section 6.3. The following diagrams show typical deck set-ups for two cases.

Program A is the main program to be compiled, and programs B and C are subprograms which have been previously compiled:



Program A is compiled and then the three binary decks are executed in order: A,B,C.

If program A, the main program, and subprogram C have been compiled, and subprogram B is to be compiled, the following deck set up can be used:



COPYBF copies the binary record containing program A onto the file named XX. Subprogram B is then compiled and placed on the load-and-go file. The programs are executed in the order specified by the control cards, the last card specifies the first program executed and the first card, the last.

A third case in which three programs are initially compiled in the same job is given in the sample subcompile program (Appendix F). A COBOL card is required to compile each program and one LGO card indicates that they will be loaded in the order compiled which is the same as the order they appear in the listing.

The table in Appendix J shows the possible binary output from a COBOL compilation.

6.3 EXECUTION

Execution of a COBOL program will follow immediately upon compilation if the binary output parameter is specified on the COBOL control card as B = LGO or if B is omitted. Binary output is placed on the load-and-go file and the object program is executed. If the binary output from compilation is placed on another file, it may be executed immediately, or at a later date, or not at all depending on whether compilation revealed the necessity for changes to the COBOL source program. Finally, the object program can be contained on a file listed in the File Name/File-Status table and may be called by name for execution at any time.

6.3.1 JOB CARD

All jobs under control of the SCOPE system must begin with a job card. This card precedes any other control cards. All control cards for a SCOPE job are placed together at the beginning of the deck and they comprise the first logical record of a job. The end of the control card record is signalled by an end of record card (7,8,9 punches in column 1). The first field of a control card begins in column 1.

The job card has the following formats:

```
n,Tt,CMfl,ECb,Pp.
```

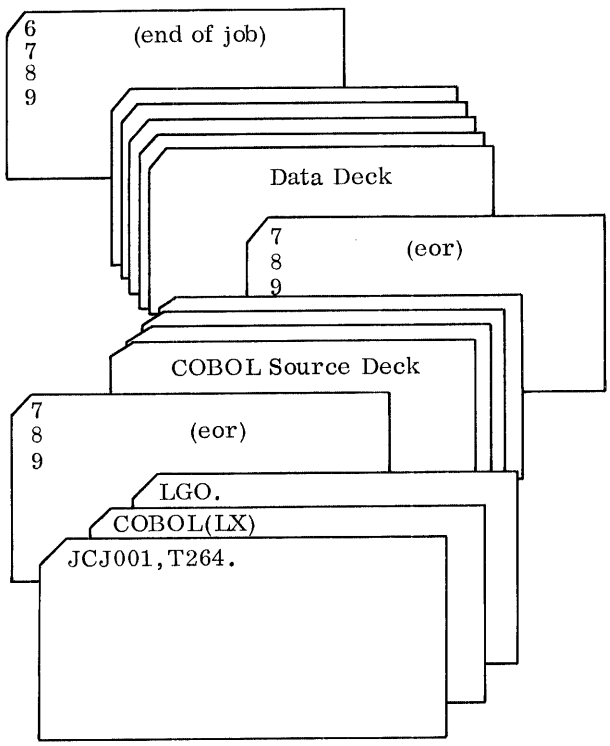
The fields are separated by commas and the last is terminated by a period. Blanks are ignored. n, must be first; others may follow in any order and are identified by initial characters. If only n is specified, installation-declared values are assumed.

Field	Description
n	Job name; 1-7 alphanumeric characters beginning with a letter. SCOPE replaces positions 6 and 7 with a unique system generated value; zero fill is provided if job name is less than 5 characters.
Tt	Total time limit for the job in seconds including compilation and execution. t = 1-5 octal digits, maximum 77777_8.

Field	Description
CM_{f1}	Field length; storage requirement for job; rounded up to a multiple of 100_8 by system; f1 = 1-6 octal digits, maximum 360,000.
ECb	Extended core storage blocks; number of ECS blocks required by the job. One block = 1000_8 words maximum. b = 1-4 octal digits
Pp	Priority level in octal at which job enters system. p is in the range $1 \leq p \leq 2^k-1$ where k is an installation provided constant ≤ 8. 1 is lowest priority.

When compilation is followed by immediate execution of the resulting object program, only the COBOL control card and the LGO card are needed following the job card.

Example:



The job named JCJ001, with a maximum time limit of approximately 3 minutes, expects the COBOL source input on INPUT file. It will compile the program and place the binary output on the load-and-go file; a listing with extended diagnostics will be placed on file OUTPUT; and if a source library is assumed, it will be on file COLIB. The job will execute the COBOL object program using the COBOL data deck provided.

6.3.2 EXECUTION CONTROL CARDS

If the program is not compiled and executed in the same job, a program call card or LOAD and EXECUTE cards are required.

LOAD card

```
LOAD(lfn)
```

This card directs the system to load the logical file named, lfn. lfn must be 1-7 alphanumeric characters; the first is alphabetic. If lfn is the system file INPUT, loading begins from the current position. Otherwise file lfn is rewound prior to loading. Loading terminates when an end of information card (6,7,8,9 punches in column 1) or an empty record is encountered (2 successive cards with 7,8,9 punches in column 1). More than one LOAD card is needed if subprograms are loaded from more than one file, for instance, if the COBOL program has overlay segments or the subcompile capability is used. The first file always determines the type of loading for all subsequent LOAD cards in the same job.

EXECUTE card

```
EXECUTE(name,p1,p2,...,pn)
```

The parameter, name, is the entry point of the program to be executed once loading is completed; if name is omitted, the system supplies the last transfer address encountered. P_i are parameters that are passed to the program to be executed. The EXECUTE card causes completion of loading. It will provide entry points from the system library if undefined references remain following execution of the LOAD card. If a program is segmented, execution begins with the main or fixed portion of the program. Subsequent segments are loaded as a result of user calls from the main program.

Program Call Card

The program call card may replace the LOAD and EXECUTE cards if the name is in the File Name/ Status Table.

name (p_1,p_2, . . . , p_n)

The program name specified is searched for in the FNT/FST table; and if found, subprograms are loaded by LOAD cards. The file is rewound before loading. The system library is searched for files not found in the FNT/FST table and the subprograms are loaded. Loading is completed and execution begun; p_i are parameters that are passed to the program to be executed.

Examples:

If a prior compilation has produced a COBOL object program on the file named ACCOUNT, the following control cards may be used to execute the program:

```
LOAD (ACCOUNT)
EXECUTE.
```

These control cards are equivalent to using the program call card: ACCOUNT.

If the file named ACCOUNT contains more than one segment, the following control cards would indicate the entry point, START, at which execution is to begin:

```
LOAD (ACCOUNT)
EXECUTE (START).
```

If a subprogram had been compiled separately and is now on a different file, this subprogram can replace a subprogram of the same name on the original file with the following cards:

```
LOAD (SUM)
ACCOUNT.
```

This will result in any subprogram on the file SUM being loaded, and any subprogram with the same name on file ACCOUNT is bypassed.

6.3.3 EQUIPMENT ASSIGNMENT

The SCOPE system assigns all files to disk unless a different assignment is specified by the REQUEST card. Any tape files used by a COBOL program must be specified on a REQUEST card. Since the control cards of a job are processed in order, a REQUEST card for a file must precede any reference to that file. The user need not request the card reader, printer, or card punch for normal input/output.

REQUEST Card

```
REQUEST,lfn,dt,dc,X.
```

This card requests the operator at the console to assign a peripheral unit to a file and describes the file and unit. The job waits for operator action before processing.

The parameter may be listed in any order. Successive blanks, commas, periods, and parentheses are ignored. If a parameter is listed more than once or is in error, a message is given and the job is terminated.

Parameter	Description
lfn	Logical file name; name used in the source program for the file. 1-7 alphanumeric characters with the first character alphabetic. This parameter may not be omitted.
dt	Device type; specifies type of device to which lfn is assigned. The form is yxx, xx may be any of the following. CP card punch LP line printer MT 1/2-inch magnetic tape LO 1/2-inch magnetic tape HI 1/2-inch magnetic tape HY 1/2-inch magnetic tape WT 1-inch magnetic tape CR card reader Dnnn disk, nnn is the disk type† If xx = MT, density on an input tape is determined by volume header label and output density is installation dependent. Density is normally 556. If the tape is unlabeled, density of 556 bpi is assumed. y has no significance if xx is other than MT, LO, HI, or HY. y may be 1, 2, or blank. If y = 2 and xx = MT, LO, HI, or HY, two units are provided and end-of-reel processing is automatic. If y = 1 or blank and xx = MT, LO, HI, or HY, the system will rewind and unload the unit when an end-of-reel condition is reached, and request a new unit.
dc	Disposition Code; special optional properties of a file are specified by dc through one of the following values: PR Print file at job termination; OUTPUT is automatically assigned. PU Punch file at job termination; PUNCH is automatically assigned. PB Punch file in binary at job termination; PUNCHB is automatically assigned. CK Check-point dump file.
X	File Declaration Code; this parameter is required if SCOPE system label processing is to be performed on 1/2-inch magnetic tape. X may = E, N, or X.

† See SCOPE manual FET description for values of nnn.

E Existing file; initial use is input. SCOPE checks the label.

N New file; initial use is output. SCOPE writes a standard volume header label.

X External file; tape file created by system other than SCOPE 3.0. X may be given for input file only. Entire record will be read and transferred to CM circular buffer without examining the last 8 characters of record for level number.

Example:

```
REQUEST, MASTER, 2MT, E.
```

The input tape file, called MASTER, has standard labels and SCOPE will check the volume header label to insure that the correct file is referenced.

6.4 COBOL SOURCE LIBRARY

Any COBOL source program that uses COPY library-name, COPY data-name FROM LIBRARY clauses, or the INCLUDE statement assumes a knowledge of the contents of the COBOL source library. The user may create the library using the COPYCL program of the SCOPE system, he may update the library with the EDITSYM program of the SCOPE system, or he may simply use a listing of the existing library. The COBOL source library consists of text decks which are distinguished from other decks by a special indicator, the number 4, in the first word of the prefix attached to all library decks. This indicator is provided by the system. The first card following the prefix cards in every COBOL text deck must contain the library-name by which this deck is referenced. The library-name starts in column 8 and may be up to 30 alphanumeric characters. It must be the only entry on the card. Anything after column 38 of the card is ignored. The remainder of the text deck contains the material to be copied or included in the source program and must be in standard COBOL source library format.

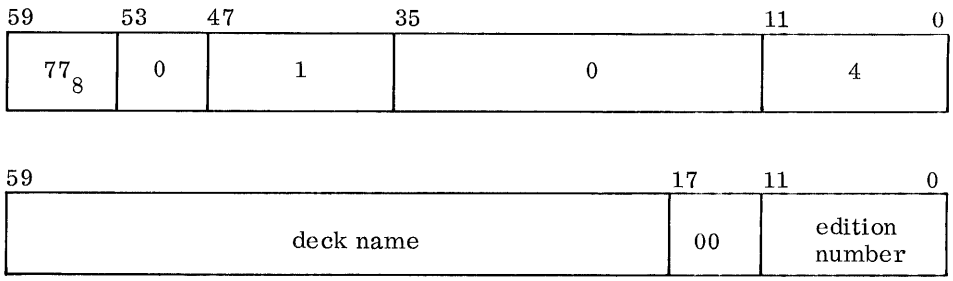
When the COPY library-name clause is used in the Environment Division Sections: SOURCE-COMPUTER, OBJECT-COMPUTER, SPECIAL-NAMES, FILE-CONTROL, and I-O CONTROL, the deck referenced by library-name will be copied into the Environment Division. Only one section may appear in a deck.

When the COPY library-name clause is used to copy an FD, SD, or RD into the Data Division, the deck referenced by library-name will be copied into the Data Division. Only one FD, SD, or RD may appear in a deck. Record descriptions may not be included in the deck.

When COPY data-name FROM LIBRARY is used in a record or item description the deck referenced by data-name will be copied into the Data Division. Only one record or item description may appear in a deck.

6.4.1 TEXT DECKS

Each file description entry, report entry, set of data items or procedures in the library is contained on a separate text deck. A COBOL library text deck is preceded by a two word prefix of the following format.



The number 4 is placed in bit positions 11-0 of the first word to indicate a COBOL library deck. The deck-name in the second word of the prefix is the name used by the system to identify the deck; it contains a maximum of 7-display code characters. The edition number is increased by one each time a new program library is requested. The remainder of the deck consists of 90 display code characters in the following format:

Column	Content
1-72	one line in source language
73-79	identifying deck name
80-84	primary sequence number
85	period if a secondary sequence number is present
86-90	secondary sequence number

The primary and secondary sequence numbers indicate the level of editing. Primary level editing permanently removes, inserts, or changes a line and the deck will be resequenced when a new program library is requested. Secondary level editing marks a card cancelled but does not actually delete it; rather it inserts a card with a secondary sequence number.

Example:

The following portion of a text deck has the library-name FILEA and contains a File Description Entry. The deck-name is COB1 and the sequence numbers are primary.

Source Image	Deck-Name	Sequence Number
FILEA		
FD REPORT-FILE	COB1	00001
LABEL RECORDS ARE STANDARD	COB1	00002
VALUE OF ID IS FILE-1	COB1	00003
DATA RECORD IS REPORT-A	COB1	00004
.	.	.
.	.	.
.	.	.

If the user wishes to change the second line and delete the third, he may use a primary or secondary edit. The primary edit will produce the following:

```
FILEA
FD      REPORT-FILE                          COB1      00001
        LABEL RECORDS ARE OMITTED.           COB1      00002
        DATA RECORD IS REPORT-A              COB1      00003
        .                                    .         .
        .                                    .         .
        .                                    .         .
```

A secondary edit produces the following:

```
FILEA
FD      REPORT-FILE                          COB1      00001
        LABEL RECORDS ARE STANDARD           COB1      00002
        VALUE OF ID IS FILE-1                COB1      00003
        LABEL RECORDS ARE OMITTED.           COB1      00003.00001
        DATA RECORD IS REPORT-A              COB1      00004
        .                                    .         .
        .                                    .         .
        .                                    .         .
```

Cards 2 and 3 are marked cancelled and left in the deck, and the new card is given a secondary sequence number. This deck will not be resequenced when a new program library is requested.

The control cards used to produce primary and/or secondary editing of a text deck are described in 6.4.3.

6.4.2 COPYCL

The COBOL source library text decks as described above are maintained on a COBOL library file. This is a standard SCOPE random access file. The program COPYCL is required to produce a COLIB file and to produce a new COBOL library file following any updating or secondary editing by the EDITSYM program. The program COPYCL is called by the following control card:

```
COPYCL(input-file-name, output-file-name)
```

The input-file-name is a standard EDITSYM library file. (If the library is being created, the EDITSYM control card will indicate this). The output-file-name is the COBOL library random access file that will contain the library in a form that the COBOL source program can reference. This file-name is also specified in the source library parameter of the COBOL control card. If no name is specified on the COBOL control card, COLIB will be the name assigned to the COBOL library. The COPYCL control card is also required when an existing COBOL source library is updated and a new program library is requested.

6.4.3 EDITSYM

The EDITSYM program must be called when a source program library is being created or maintained. The EDITSYM control cards used to edit an existing deck are described below. The EDITSYM program is called by the following SCOPE control card:

```
EDITSYM(p1,p2,p3,p4,p5)
```

Parameters are free field.

Parameter	Meaning	Description	
I	Correction Input	absent	corrections on INPUT
		I	corrections on INPUT
		INPUT	corrections on INPUT
		I = lfn	corrections on lfn
		INPUT = lfn	corrections on lfn
C	Compile	absent	no compile output
		C	compile output on COMPILE
		COMPILE	compile output on COMPILE
		C = 0	no compile output
		COMPILE = 0	no compile output
		C = lfn	compile output on lfn
		COMPILE = lfn	compile output on lfn
L	List	absent	no list
		L	list on OUTPUT
		LIST	list on OUTPUT
		L = 0	no list
		LIST = 0	no list
		L = lfn	list on lfn
		LIST = lfn	list on lfn
OPL	Old Program Library	absent	no old program library
		OPL = 0	no old program library
		OPL	old program library on OPL
		OPL = lfn	old program library on lfn
NPL	New Program Library	absent	no new program library
		NPL = 0	no new program library
		NPL	new program library on NPL
		NPL = lfn	new program library on lfn

New decks are introduced by the following control card:

```
*DECK,dn,4
```

This card is placed directly in front of a COBOL text deck to be inserted in the library. dn is the deck name that identifies this deck in the deck prefix and in columns 73-80 of the card image. The number 4 identifies this deck as a COBOL source deck.

A text deck is terminated by the control card:

```
*END
```

An entire text deck or set of text decks may be copied from the program library onto the compile file with the following control card:

```
*COPY,p1,p2
```

The parameters may be deck names or an asterisk. If p_1 is a deck name copying begins with the deck identified by that name and continues up to and including p_2. If p_2 is not specified, only the deck identified by p_1 is copied. If p_1 is an asterisk, copying begins at the present position and continues through p_2. If p_2 is an asterisk, copying begins with p_1 and continues to the end of the program library.

Common and text deck names are listed in the order they appear in an old or new program library with the following control card:

```
*CATALOG,lfn
```

lfn is the name of the old or new program library. All common and text deck names contained on lfn are listed on OUTPUT.

Edit Control Cards

An existing library deck may be edited. The control cards specify the card to be altered, or deleted, or the point where new cards are to be inserted.

The EDIT card must be specified last in the set of control cards to modify a deck. It specifies the deck name of the text deck to which the sequence numbers refer.

```
*EDIT,dn
```

dn is the deck name of the deck to be edited.

Editing may be done at the primary or secondary level. *INSERT, *DELETE, and *RESTORE are primary level edit cards; *CANCEL, and *ADD are secondary level edit cards.

```
*INSERT,n
```

Correction cards are inserted in the deck following the card with primary sequence number (integer n). The corrections are terminated by the *EDIT card. Inserted cards are primary text and the text deck will be resequenced if a new library is requested.

```
*DELETE,m,n
```

The portion of the text with primary sequence numbers m through n, inclusive are deleted. m and n must be integers. If n is omitted, only card m is deleted. Source cards may follow the DELETE card; they will be inserted into the deck following the last deleted card. Deletions and insertions are primary corrections, deleted cards are removed and the remaining cards are resequenced when the new program library is requested.

```
*RESTORE,m,n
```

When a deck is altered by secondary level editing, the old cards remain in the deck with their original primary sequence numbers. This card is used to restore the portion of a deck, m through n inclusive, that has been altered by secondary editing. All cards within the range of m through n that have been canceled as a result of secondary editing are restored as normal primary text cards; all added secondary text cards within the range m through are removed.

Secondary level editing is accomplished with the following cards:

```
*CANCEL,mn
```

m and n may be of form j.k where j is a primary number, and k is a secondary sequence number. All cards are canceled inclusive between m and n. Primary cards are marked canceled but not removed; secondary cards are removed. Source cards may follow the *CANCEL control card and are inserted after the last canceled card. The insertions are marked as secondary text. Cancelation does not cause re-sequencing of the primary cards when a new program library is requested.

```
*ADD,n
```

This is a secondary editing control card; n may be of the form j.k as defined above. Ensuing cards are inserted as secondary text. Addition does not result in re-sequencing of primary cards when a new program library is requested.

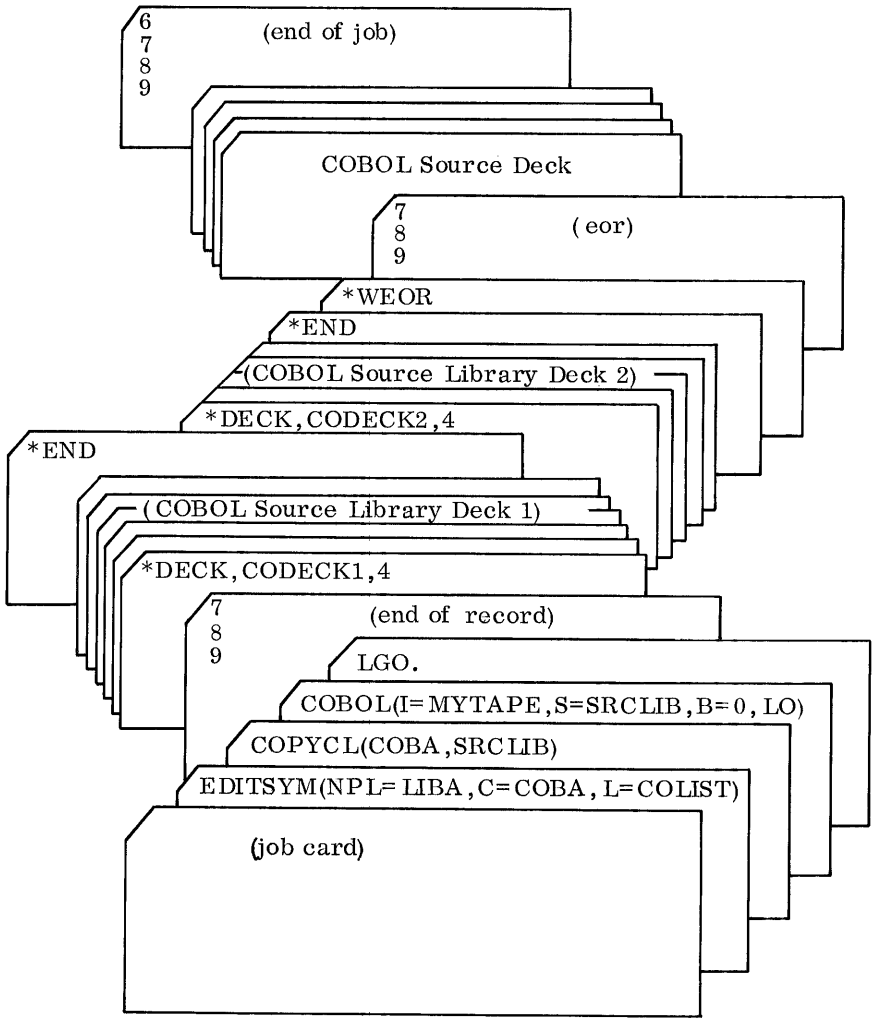
An additional EDITSYM control card allows the user to compile or assemble specific decks from different files:

```
*COMPILE,lfn
```

*COMPILE, lfn allows the user to write a compile file on lfn. When this card is read from the correction input, the compile file for the deck specified on the following *EDIT, *DECK, or *COPY card is written on lfn. Once a *COMPILE card has been encountered, compile files for all remaining text decks must be requested by a *COMPILE control card.

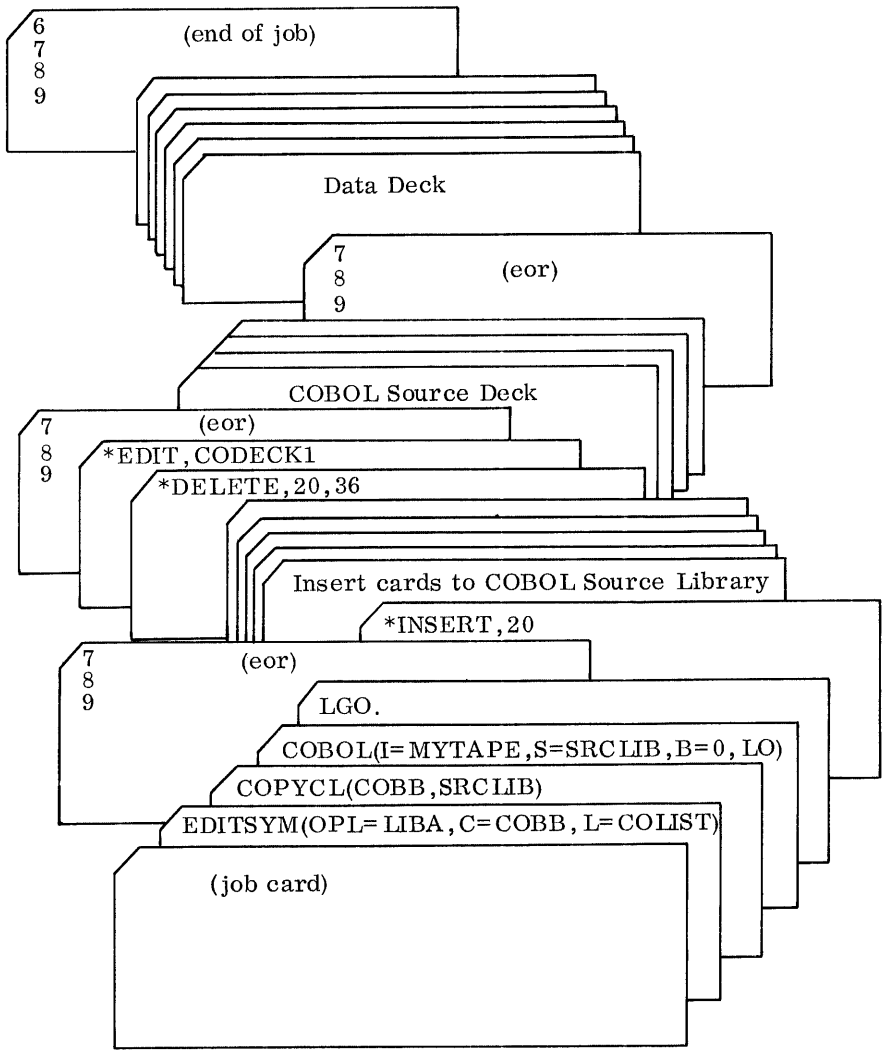
Example:

Typical deck setup for compilation with new COBOL source library decks:



The EDITSYM program places the two COBOL source library decks on the library file LIBA, prepares a compile file COBA, and a listing of the library on the print file COLIST. The COPYCL program prepares a random access file SRCLIB for use by the COBOL compiler from the library file COBA produced by EDITSYM. The COBOL control card indicates to the COBOL compiler that this library will be found on SRCLIB.

Should the user wish to change the text deck named CODECK1, the deck setup for altering and preparing a new COBOL library file from the altered decks would be:



This deck setup updates the first COBOL text deck CODECK1 on the program library file LIBA and prepares a new compile file, COBB. Since editing is done on the primary sequence numbers, the new program library file is resequenced, and the omitted cards are permanently deleted rather than canceled. COPYCL puts the new deck on the special random access file SCRLIB for use by the COBOL compiler. This particular deck setup runs with a deck of data for use by the COBOL program. A listing of the updated COBOL source library is again output on COLIST.

APPENDIX SECTION

THE COBOL LANGUAGE

A

This appendix contains a general description of the COBOL language. It contains the following information:

1. The COBOL character set
2. Words

   User-defined words:

   - data-names
   - procedure-names
   - condition-names
   - qualifiers
   - literals

   COBOL words:

   - Reserved Words
   - Key Words
   - Optional Words
   - Figurative Constants
   - Connectives and Separators
3. Punctuation
4. Rules for using the COBOL Coding Sheet

The COBOL Character Set

The COBOL language contains a set of characters which the programmer may combine according to specified rules to form the names and values for the source program. The COBOL character set consists of the numerals 0 through 9, the 26 letters of the alphabet, and the following special characters:

Blank or Space	$ Dollar Sign
+ Plus Sign	, Comma
- Minus Sign or Hyphen	. Period or Decimal Point
* Asterisk	" Quotation Mark
/ Stroke or Slash	(Left Parentheses
= Equal Sign	) Right Parenthesis

Some of these characters have additional uses, as follows:

Characters used in Arithmetic Expressions

+ Addition	* Multiplication
- Subtraction	/ Division

Characters used in Relations

= Equality

< Less than

> Greater than

Characters used in Punctuation

" Quotation Mark	Blank or Space
(Left Parenthesis	. Period
) Right Parenthesis	, Comma

Characters used in Editing

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

WORDS

Words are either specified by the user or they are COBOL words. In the COBOL format, COBOL words are represented by full capitalization. User specified words include all names assigned by the user to elements in the program, and they must never be from the set of COBOL reserved words (Appendix C). User-defined names include:

data-names	identifiers
procedure-names	file-names
condition-names	mnemonic-names
library-names	

Literals and named constants are also considered to be user specified words. Rules are given below.

Data-Names

Data-names are formed by combining any of the following characters:

0, 1, . . . , 9

A, B, . . . , Z

- (the hyphen)

A data-name may contain up to 30 of these characters with no imbedded blanks. Hyphens may be used freely within the name except they may not be the first or last character in a name, or used consecutively. All data-names must contain at least one alphabetic character; it need not be the first character unless explicitly stated.

Examples:

QUANTITY-ON-HAND	MESSAGE
LAST-YEAR-PROFIT	100A
ITEM-NUMBER	FILE-1

Procedure-Names

Procedure-names identify a paragraph or section in the Procedure Division. Procedure-names are formed the same as data-names except that they may consist exclusively of numeric characters. Numeric names are equivalent only if they have the same number of digits and the same value; 0023 is not equivalent to 023. All procedure-names must be unique in themselves or be made unique by qualification. Procedure-names are terminated by a period or the word SECTION, an optional priority number, and a period.

Examples:

```
PROC-1.                    A-INPUT SECTION.
002.                       100 SECTION 55.
```

Condition-Names

Condition-names are assigned to the values an item may assume. A condition-name is formed the same as a data-name, but it must always be preceded by the level number 88.

Example:

```
03 GRADE
        88 GRADE-ONE VALUE IS 1.
        88 GRADE-TWO 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 13 THRU 99.
```

File-Names/Implementor Names

File-names assigned to a file in the File Description entry of the Data Division are formed exactly like data-names. Files may also be specified as implementor names in the Environment Division. Implementor names are slightly different since they must conform to SCOPE system standards. The first seven characters must be unique, the first character must be alphabetic, and no hyphen is allowed within the first seven characters.

Qualifier

Every name in a source program must be unique, either because no other name has the identical spelling, or because the name exists within a hierarchy of names, such that it can be made unique by mentioning one or more of the higher levels of the hierarchy. The higher levels are called qualifiers when used in this way. The qualification process is performed by writing IN or OF after the name followed by a qualifier. The choice between IN or OF is based on readability; they are logically equivalent. Only sufficient qualification must be mentioned to make the name unique. Whenever the data item or paragraph is referenced, any necessary qualifiers must be written as part of the name.

Identifier

Identifiers are used throughout the Procedure Division to reference items defined in the Data Division. Identifiers are data-names followed by the qualifiers or subscripts needed to make reference to the item unique. If no qualification or subscripting is required, the identifier is a simple data-name.

Examples:

```
DAY OF MASTER-DATE
FIRST IN GRADE-ONE
MALE-FEMALE(2,5,1)
```

Literal

A literal is an explicit specification of the value used in the object program. Literals are numeric and non-numeric. A numeric literal is any combination of 0 through 9; it may be preceded by a plus or minus sign, and may contain a decimal point. The decimal point may not be the rightmost character. If no sign is specified, the literal is positive.

Examples of numeric literals:

```
15067893251459
-12572.6
+25675
1435.89
```

A non-numeric literal may be any character from the COBOL set, including blanks. A non-numeric literal must be enclosed in quotation marks. The quotation marks are not part of a non-numeric literal, and may not be used in forming such a literal.

Examples of non-numeric literals:

```
"LINE-COUNTER"
"PAGE 144 IS MISSING"
" A B C D E F "
"-125.56"
"PAGE NUMBER IS      "
```

The literal "-125.56" is not the same as the literal -125.56; the quotation marks make it a non-numeric literal and prevent it from being used for computation.

Named Constants

A name may be assigned to a specific value and used wherever the value is required. This is a named constant; it is used like any other data name except that it refers to a constant value rather than to a value that varies during processing.

COBOL Reserved Words

The COBOL language contains a set of reserved words which have particular significance to the processor, and which may be used only in the presented manner. They may not be used as data names, procedure names, condition names, and so forth. The two categories of reserved words are optional words and key words.

The notation used in this manual distinguishes between COBOL optional and key words in the following manner: All COBOL reserved words are printed entirely in capital letters; the key words are underlined, the optional words are not.

Optional COBOL words may be included in the source program to improve readability. They are recognized by the processor but are ignored since they are not needed to compile the object coding. Some words are optional in certain situations but not in others. An optional word must be correctly spelled. When an optional word is omitted, it may be replaced only by a word explicitly stated to be its equivalent.

Key words, on the other hand, are those COBOL words which are essential to convey the meaning of a clause or statement. A key word may not be omitted. All verbs, for example, are key words which must be included to designate the operation to be performed.

Examples:

A IS GREATER THAN B

A IS GREATER B

A GREATER THAN B

A GREATER B

All these expressions are correct and have the same meaning whether or not the optional words IS and THAN are used.

Figurative Constants

Certain names in the COBOL set of reserved words represent constants. The constant values associated with these names are known as figurative constants. The singular and plural forms of the names are equivalent and may be used interchangeably.

The names and class associated with the figurative constants are listed below. They generate strings of homogeneous information, the length of which is determined by the size of the receiving item. A figurative constant may be used wherever a literal of the same class is indicated in the COBOL format.

Name	Class	
ZERO ZEROS ZEROES	Numeric	In computations, all three figurative constants represent the numeric value zero. Used any other way, they represent a series of zero characters.
SPACE(S)	Alphanumeric	Either word represents a series of spaces (blanks).
HIGH-VALUE(S) UPPER-BOUND(S)	Alphanumeric	Any of the words represents a series of the character with the highest value in the COBOL display code collating sequence.
LOW-VALUE(S) LOWER-BOUND(S)	Alphanumeric	Any of the words represents a series of the character with the lowest value in the COBOL display code collating sequence.
RECORD-MARK	Alphanumeric	A special character which indicates the end of a logical record.
QUOTE(S)	Alphanumeric	Either word represents a series of the quotation mark character. The word QUOTE may not be substituted for the symbol " enclosing a non-numerical literal.
ALL any-literal	Alphanumeric or Numeric	The word ALL followed by any literal results in a sequence of all characters comprising the literal repeated in the order in which they occur. If used in a move operation, or a conditional statement, the literal, numeric or non-numeric, must be enclosed in quotation marks; it is always alphanumeric. Used otherwise (EXAMINE) only a non-numeric literal need be enclosed in quotation marks; ALL any-literal, in this case, could be either numeric or alphanumeric. ALL followed by a figurative constant is redundant.

Examples of the use of figurative constants:

MOVE QUOTES TO AREA-A	Assuming AREA-A consists of five character positions, this statement moves the configuration """""" to AREA-A.
DISPLAY QUOTE "NAME" QUOTE	This statement results in "NAME" being displayed.

MOVE SPACES TO TITLE	The item named TITLE is set to all spaces (or blanks).
MOVE ALL "4" TO COUNT-FIELD	Assuming COUNT-FIELD has a picture of X(4), a 4 is placed in each position of the item named COUNT-FIELD.
IF ALL "4" EQUALS COUNT-FIELD...	Assuming COUNT-FIELD is a conditional item with a picture of 9(4) or X(4), this compares 4444 with the value of COUNT-FIELD.
MOVE ZEROS TO REGISTER	This places 0 in each position of the item named REGISTER.
MOVE ALL "NO-OP" to EMPTY	Assuming EMPTY consists of 12 character positions, EMPTY is filled with repetitions of the characters of the non-numeric literal, NO-OPNO-OPNO.

Special Registers

Special registers used for particular purposes within a COBOL program are defined by the COBOL words:

TALLY

PAGE-COUNTER

LINE-COUNTER

TALLY is a Computational-1 item automatically generated in a special Common-Storage Section so that it will be available to all segments of a COBOL program. Its size is eight digits. TALLY is used with the TALLYING option of the EXAMINE statement to receive the numbers resulting from execution of the statement.

PAGE-COUNTER is a fixed data name used as a page counter by the Report Writer. It is automatically generated as part of the Report Section for use as a source item to present page numbers within a report group. One page counter is supplied for each report if the word PAGE-COUNTER is specified in the SOURCE item of a report description.

LINE-COUNTER is a fixed data-name used as a line counter by the Report Writer. It is automatically generated as part of the Report Section for use in determining any page or overflow HEADING and FOOTING report groups. One line counter is supplied for each report if a PAGE LIMIT clause is included in the Report Description Entry.

Connectives

Connective words are used to connect series of names or expressions indicating the relationship between the elements. The logical connectives AND and OR join simple conditional expressions to form compound conditional expressions. The truth or falsity of compound conditional expressions

will depend not only on the truth or falsity of the constituent simple conditional expressions, but also on the particular logical connective used.

Series Separators

Two or more words written in a series must be separated by a space; they may be separated by the following separators:

AND , (comma) , AND (comma followed by space and AND)

The following separators may be used between statements in a sentence to improve readability:

THEN ; (semicolon) ; THEN (semicolon followed by space and THEN)

Any of the above separators may be used between items, but may not be followed immediately by another such separator.

Punctuation

Most punctuation marks are optional in COBOL. The inclusion or omission of a comma does not affect the compilation process. The general rule concerning punctuation marks is as follows:

When a period, semicolon, or comma is used, it immediately follows a word; but it must be followed by at least one space before the first character of the next word is specified. In a non-numeric literal, the beginning quotation mark may not be followed by a space, nor may the ending quotation mark be preceded by a space, unless such spaces are required as part of the literal.

Left and right parentheses specify the order in which conditional and arithmetic expressions are to be evaluated. A left parenthesis may not be immediately followed by a space unless it is followed by +, -, or another left parenthesis, in which case a space is required; nor may a right parenthesis be immediately preceded by a space unless it is preceded by a right parenthesis. Parentheses must be specified in pairs. The same punctuation rules apply to words contained in parentheses as to separate words. Punctuation marks after the right parenthesis are followed by at least one space before the left parenthesis of the next expression.

The COBOL Coding Sheet

The specifications for the source program are written on COBOL coding sheets according to the formats contained in this manual.

All division names, section names, and paragraph names start in column 8 of the coding sheet. Division names are followed by a period and the rest of the line must be blank. Section names are followed by the word SECTION and a period. If priority is specified, it follows the word SECTION before the period. The remainder of the line is blank, except if the section is a DECLARATIVE, it may be followed by a USE statement. Paragraph names are followed by a period and at least one space. The text may follow or may start at column 12 of the next line.

The level indicators FD, SD, RD and 01 begin in column 8, they are followed by one or more spaces and the associated entry. All other level indicators begin in or after column 12 followed by a space and associated entry.

Sequence numbers, if specified, are in columns 1 through 6. Program identification is placed in columns 73 through 80. Lines may be broken at any convenient point, spaces may remain at the end of the line. When a word or a numeric literal is split between two lines, a hyphen must be specified in column 7 of the second line. If a non-numeric literal is split between two lines, a quotation mark must be specified in or after column 12 of the second line in addition to a continuation hyphen in column 7 of the second line. In this case only, the blanks at the end of the first line are considered part of the literal.

Rules for using the COBOL coding sheet are summarized below:

Element	Type	Division	Column Position	Remarks
Name	Division-name	ALL	8	Name must be followed by a period; remainder of the line must be blank.
	Section-name	ENVIRONMENT DATA PROCEDURE	8	Name must be followed by a space, the word SECTION, priority if specified, and a period; remainder of the line must be blank, or contain a USE sentence.
	Paragraph-name	IDENTIFICATION ENVIRONMENT PROCEDURE	8	Name must be followed by a period and at least one space. Text may follow on same line or at column 12 on next line.
Data Description Entry	File Description Sort Description Report Description	DATA	8	Descriptions begin with level indicator, FD, SD, or RD, two or more spaces separate it from data name. Clauses are separated by one or more spaces.
	Record Description	DATA	At 8 or after 11, depending on the identification.	Same as file description entry. Level number 01-49, 66, 77 and 88. Only 01 entries may begin in column 8.
Sentence	First sentence of a paragraph or section	IDENTIFICATION ENVIRONMENT PROCEDURE	Following period and 1 space after paragraph name or section name, or on next line at or after column 12.	
	All other sentences	IDENTIFICATION ENVIRONMENT PROCEDURE	Following period and 1 space after the previous sentence.	Sentences may be written in columns 12 through 72 only.
Continued Elements	Data description entry	DATA	At or after 12.	Line breaks may occur at any convenient point, with spaces at end of line if desired. If a word or literal is split between two lines, a hyphen must be specified in column 7 of the second line.
	Sentence	IDENTIFICATION ENVIRONMENT PROCEDURE	At or after 12.	
Non-Program Entry	Sequence Number	ALL	1-6	Sequence number does not affect the object program; processor does check for correct sequencing.
	Program Identification	ALL	73-80	Identification information does not affect object program.

COBOL DISPLAY CODE AND COLLATING SEQUENCE B

The following table shows the relationship between the characters of the COBOL set and their equivalent machine, printer, tape, or card representation. They are listed according to ascending collation sequence.

Characters shown with an asterisk are not available in COBOL source language but would be treated in the sequence indicated if present in data. They are part of the COBOL character set.

Collating Sequence	Display Character	Internal Representation	BCD Tape	Hollerith Punch
00	Δ	55	20	space bar
01	≤	74	15	8-5
02	¢ *	61	17	8-7
03	→ *	65	35	-
04	≡ *	60	36	0-8-6
05	∧ *	67	37	0-8-7
06	↑ *	70	55	11-8-5
07	↓ *	71	56	11-8-6
08	> *	73	57	11-8-7
09	≥ *	75	75	12-8-5
10	¬ *	76	76	12-8-6
11	. (period)	57	73	12-8-3
12	)	52	74	12-8-4
13	; *	77	77	12-8-7
14	+	45	60	12
15	$	53	53	11-8-3
16	*	47	54	11-8-4
17	-	46	40	11
18	/	50	21	0-1
19	, (comma)	56	33	0-8-3
20	(	51	34	0-8-4
21	=	54	13	8-3

Collating Sequence	Display Character	Internal Representation	BCD Tape	Hollerith Punch
22	≠	64	14	8-4 †
23	< *	72	72	12-0 or 12-8-2
24	A	01	61	12-1
25	B	02	62	12-2
26	C	03	63	12-3
27	D	04	64	12-4
28	E	05	65	12-5
29	F	06	66	12-6
30	G	07	67	12-7
31	H	10	70	12-8
32	I	11	71	12-9
33	∨ *	66	52	11-0 or 11-8-2
34	J	12	41	11-1
35	K	13	42	11-2
36	L	14	43	11-3
37	M	15	44	11-4
38	N	16	45	11-5
39	O	17	46	11-6
40	P	20	47	11-7
41	Q	21	50	11-8
42	R	22	51	11-9
43	]	62	32	0-8-2 ††
44	S	23	22	0-2
45	T	24	23	0-3
46	U	25	24	0-4
47	V	26	25	0-5
48	W	27	26	0-6
49	X	30	27	0-7
50	Y	31	30	0-8
51	Z	32	31	0-9

† This is the COBOL quotation mark.

†† This is the COBOL record mark.

Collating Sequence	Display Character	Internal Representation	BCD Tape	Hollerith Punch
52	:	63	00	8-2
53	0	33	12	0
54	1	34	01	1
55	2	35	02	2
56	3	36	03	3
57	4	37	04	4
58	5	40	05	5
59	6	41	06	6
60	7	42	07	7
61	8	43	10	8
62	9	44	11	9
—	% or ?	0046	16	8-6
—	—	00	—	ESCAPE

The ESCAPE punch defines an escape to the display code following '00'. '00' may ESCAPE to any code except '00' which defines an end of line or card. The only codes defined are:

0000 - end of line

0046 - % or ? (depending on printer)

COBOL RESERVED WORD LIST

The reserved words of the COBOL language are listed below. They have preassigned meanings and are interpreted in a particular way by the COBOL processor. They must be used only as described in this manual. Words preceded by an asterisk represent an unimplemented feature not described in this manual. Both key words and optional words are listed, since certain words are key words in some contexts and optional in others, no distinction is made in the list.

ABOUT
ACCEPT
ACCESS
ACTUAL
ADD
*ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALPHANUMERIC
ALTER
ALTERNATE
AN
AND
*APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR

BEFORE
BEGINNING
BINARY
*BITS
BLANK
BLOCK
BY

CF
CH
CHARACTERS
CHECK
CLASS
*CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COMMA
COMMON-STORAGE
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTE
CONFIGURATION
CONSOLE
CONSTANT
CONTAINS
CONTROL
CONTROLS
*CONVERSION
COPY
CORRESPONDING
CURRENCY

DATA
DATE-COMPILED
DATE-WRITTEN
DE
DECIMAL
DECIMAL-POINT
DECLARATIVES
*DEFINE
DENSITY
DEPENDING
DESCENDING
DETAIL
DIGITS
DISPLAY
DIVIDE
*DIVIDED
DIVISION
DOLLAR
*DOWN

EDITION-NUMBER
ELSE
END
ENDING
ENTER
ENVIRONMENT
EQ
EQUAL
EQUALS
ERROR
EVERY
EXAMINE
EXCEEDS
EXIT
*EXPONENTIATED

FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILLER
FINAL
FIRST
FLOAT
FOOTING
FOR
*FORMAT
FROM

GENERATE
GIVING
GO
GQ
GR
GREATER
GREATER-EQUAL
GROUP

*HASHED
HEADING
HIGH
HIGH-VALUE
HIGH-VALUES
*HOLD
HYPER

ID
IDENTIFICATION
IF
IN
INCLUDE
*INDEX
*INDEXED
INDICATE
INITIATE
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
INVALID

I-O
I-O-CONTROL
IS

JUSTIFIED

KEY
*KEYS

LABEL
LAST
LEADING
LEAVING
LEFT
LESS
LESS-EQUAL
LIBRARY
LIMIT
LIMITS
LINE
LINE-COUNTER
LINES
LOCATION
LOCK
LOW
LOW-VALUE
LOW-VALUES
*LOWER-BOUND
*LOWER-BOUNDS
LQ
LS

*MEMORY
*MINUS
MODE
*MODULES
MOVE
MULTIPLE
*MULTIPLIED
MULTIPLY

NEGATIVE
NEXT
NGR
NLS
NO
NOT
NOTE
NQ
NUMBER
NUMERIC

OBJECT-COMPUTER
*OBJECT-PROGRAM
OCCURS
OF
OFF
OH
OMITTED
ON
OPEN
OPTIONAL
OR
OTHERWISE
OUTPUT
OV
OVERFLOW

PAGE
PAGE-COUNTER
PERFORM
PF
PH
PICTURE
PLACES
PLUS
POINT
POSITION
*PREPARED
PRIORITY
PROCEDURE
PROCEDURES
PROCEED
*PROCESS
PROCESSING
PROGRAM-ID
PROTECT
PUNCH
PUNCHB

QUOTE
QUOTES

RANDOM
*RANGE
RD
READ
RECORD
RECORD-MARK
RECORDING
RECORDS
REDEFINES
REEL
REEL-NUMBER
RELEASE
REMARKS
RENAMES
RENAMING
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETENTION-CYCLE
RETURN
REVERSED
REWIND
RF
RH
RIGHT
ROUNDED
RUN

*SA
SAME
SD
*SEARCH
SECTION
SECURITY
SEEK
*SEGMENT-LIMIT
SELECT
SELECTED
SENTENCE
SEQUENCED
SEQUENTIAL
*SET
SIGN
SIGNED
SIZE
SORT
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STATUS
STOP
SUBTRACT
SUM
*SUPERVISOR
SUPPRESS
SWITCH
*SYMBOLIC
SYNCHRONIZED

TALLY
TALLYING
TAPE
TERMINATE
THAN
THEN
THROUGH
THRU
TIMES
TO
TYPE

UNEQUAL
*UP
*UPPER-BOUND
*UPPER-BOUNDS
UNIT
UNTIL
UPON
USAGE
USE
USING

VALUE
VALUES
VARYING

WHEN
WITH
*WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

INTERMEDIATE RESULTS IN ARITHMETIC EXPRESSIONS D

COMPUTATIONAL-2

The operations specified in an arithmetic expression or statement is accomplished in single-precision rounded floating point (COMPUTATIONAL-2) for all operations succeeding the first encounter of a COMPUTATIONAL-2 operand item or exponentiation in the arithmetic expression.

When all result items are COMPUTATIONAL-2, all computation is performed in COMPUTATIONAL-2.

COMPUTATIONAL and COMPUTATIONAL-1

When all operand items are COMPUTATIONAL or COMPUTATIONAL-1 and at least one of the results is COMPUTATIONAL or COMPUTATIONAL-1, and the arithmetic expression contains no exponentiation, computation is performed either in COMPUTATIONAL or COMPUTATIONAL-1, usually COMPUTATIONAL-1. Either single or double precision will be used to obtain up to 14 or 28 decimal digits of precision to accommodate the size of the intermediate sum, product, or quotient.

When more than 28 decimal digits of precision would be required, loss of accuracy may occur in the intermediate result. Outlined below is the method by which the compiler determines the size and decimal point location of the intermediate results.

The size and point location for an intermediate result is computed from the size and point location of the two operands. One of the operands may be the result of a previous operation of the same COMPUTE statement.

If the intermediate result field exceeds 28 digits, an extended diagnostic is printed at compute time and at object time if all 28 digits are actually used; the result is truncated by dropping binary bits.

The following formulas are used in computing the size and point location for an intermediate result.

Let S_1 be the size of the operand 1 in number of digits and let P_1 be the point location or the number of decimal places. Let S_2 and P_2 be the size and point location of operand 2.

If the operation is plus (+) or minus (-), the size of the sum is:

$$S_t = \max((S_1 - P_1), (S_2 - P_2)) + \max(P_1, P_2)$$

The point location is:

$$P_t = \max(P_1, P_2) *$$

* When a series of operands is to be added or subtracted, the compiler will increase the size to take care of any possible carry.

If the operation is multiply (*), the size of the product is:

$$S_t = S_1 + S_2$$

The point location is:

$$P_t = P_1 + P_2$$

If the operation is divide (/), the size of the quotient is:

$$S_t = S_1 + S_2$$

The point location is:

$$P_t = P_2 + S_1 - P_1$$

where operand 1 is the divisor.

Example:

Item	PICTURE	Value
A	9(5)V9(5)	Unknown
B	9(11)	500.
C	9(6)	11.
D	9(3)	333.
E	V99	0.03
F	PPP99	.0007
G	9(5)	1.

COMPUTE A = B*(((C/D)/E) + F/G)

The processing steps would be:

1. $C/D \rightarrow T_1$ (first intermediate result, T_1)
2. $T_1/E \rightarrow T_2$ (second intermediate result, T_2)
3. $T_2 \rightarrow$ Temp-Cell-1 (first temporary storage cell)
4. $F/G \rightarrow T_3$
5. $T_2 + T_3 \rightarrow T_4$
6. $B*T_4 \rightarrow A$

Size and point location would be determined as follows:

1. C/D $\rightarrow T_1$ method C

   $S_C = 6$ $S_D = 3$

   $P_C = 0$ $P_D = 0$

   $S_{T_1} = S_1 + S_2 = S_C + S_D = 6 + 3 = 9$

   $P_{T_1} = P_2 + S_1 - P_1 = P_C + S_D - P_D = 0 + 3 - 0 = 3$

   The intermediate picture would be: 9(6)V9(3)

2. T_1/E $\rightarrow T_2$ method C

   $S_{T_1} = 9$ $S_E = 2$

   $P_{T_1} = 3$ $P_E = 2$

   $S_{T_1} = S_1 + S_2 = S_{T_1} + S_E = 9 + 2 = 11$

   $P_{T_1} = P_2 + S_1 - P_1 = P_{T_1} + S_E - P_E = 3 + 2 - 2 = 3$

   The intermediate picture would be: 9(8)V9(3)

3. T_2 $\rightarrow$Temp-Cell-1 no scaling is involved.

4. F/G $\rightarrow T_3$ method C

   $S_F = 2$ $S_G = 5$

   $P_F = 5$ $P_G = 0$

   $S_{T_3} = S_1 + S_2 = S_F + S_G = 2 + 5 = 7$

   $P_{T_3} = P_2 + S_1 - P_1 = P_F + S_G - P_G = 5 + 5 - 0 = 10$

   The intermediate picture would be: PPP9(7)

5. $T_2 + T_3 \rightarrow T_4$ method A

   $S_{T_2} = 11$ $S_{T_3} = 7$

   $P_{T_2} = 3$ $P_{T_3} = 10$

   $S_t = \max((S_1 - P_1), (S_2 - P_2)) + \max(P_1, P_2) + 1$ (see footnote, p. D-1)

   $= \max((S_{T_2} - P_{T_2}), (S_{T_3} - P_{T_3})) + \max(P_{T_2}, P_{T_3}) + 1$

$$= \max((11-3), (7-10)) + \max(3, 10) + 1$$

$$= \max(8, -3) + \max(3, 10)$$

$$= 8 + 10$$

$$= 18$$

$$P_t = \max(P_1, P_2) = \max(P_{T_2}, P_{T_3}) = \max(3, 10) = 10$$

The intermediate picture would be: 9(8)V9(10)

6. $B*T_4$: method B

$S_B = 11$ $\quad S_{T_4} = 18$

$P_B = 0$ $\quad P_{T_4} = 10$

$$S_t = S_1 + S_2 = S_B + S_{T_4} = 11 + 18 = 29$$

$$P_t = P_1 + P_2 = P_B + P_{T_4} = 0 + 10 = 10$$

The resulting picture would be: 9(19)V9(10); however, since this is the final result, the picture given for A, 9(5)V9(5), is used. When the move to A is executed, the assumed decimal points are aligned resulting in a loss of 14 high order digits plus 5 fractional digits to the value placed in A.

If there were yet another step in the computation, the intermediate picture could still result in a loss of digits due to the size limitation for intermediate results. The intermediate picture would be: 9(19)V9(10), but because low order binary bits may possibly be dropped at object time, approximately one decimal place might be lost.

Using the data set forth in the original example, the following steps would take place (parentheses indicate contents of location):

		Result	PICTURE	Contents
Step 1: (C) / (D) = 11/333.	=	.03303	T_1 = 9(6)V9(3)	.033
Step 2: (T_1) / (E) = .033/.03	=	1.1	T_2 = 9(8)V9(3)	1.100
Step 3: T_2 → Temp-Cell-1 No computation involved				
Step 4: (F) / (G) = .0007/1.	=	.0007	T_3 = PPP9(7)	.0007000000
Step 5: (T_2) + (T_3) = 1.100 + .0007	=	1.1007	T_4 = 9(8)V9(10)	1.1007000000
Step 6: (B)*(T) = 500. × 1.1007000000	=	550.3500000000	A = 9(5)V9(5)	00550.35000

Temporary Result Field

A maximum of 9 reusable fields are provided for storage of intermediate results.

Use of Arithmetic Expressions in Comparisons

When no result field is defined, as in a comparison, the method of computation and the size and point location of both the intermediate and final result are determined by the rules given above for intermediate results.

CALLING SEQUENCE FOR THE ENTER STATEMENT E

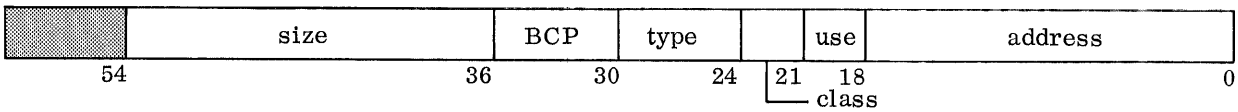
The ENTER subroutine-name statement generates the following sequence:

```
    SA1     location of parameter list
+   RJ      subroutine-name
```

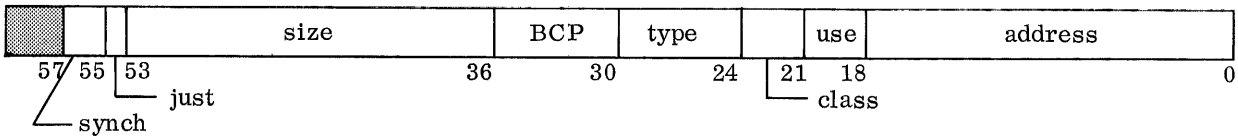
The parameter list consists of one full word for each data-name, procedure name, or file name specified in the source statement. The list is terminated by a full word of binary zeros. The location of the parameter list is followed by a return jump to the subroutine referenced by the ENTER statement. Parameter formats are shown below. There are four possible formats for data-name depending on whether it is a group item or, if not, is a computational-2, numeric, or non-numeric elementary item.

Data-Name Formats

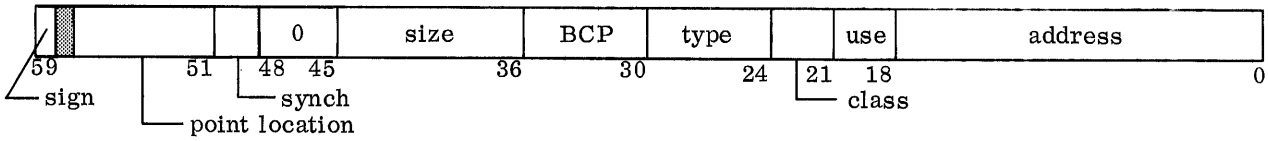
Group Name



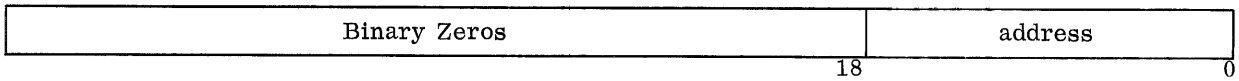
Non-Numeric Item-name



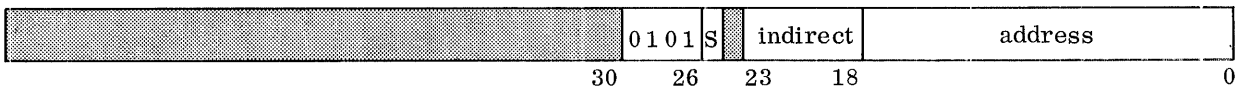
Numeric Item-name



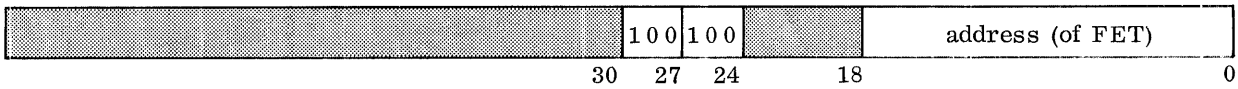
COMPUTATIONAL-2 Item



Procedure-name



File-name



The parameter fields from right to left are defined as follows:

Address — This field is always in bit positions 17-0. It is the address of the data-name item, or the file environment table (FET) for a file-name, or it is the address of a procedure or an indirect reference to this procedure.

Indirect — If this field is non-zero, the procedure address is indirect; it points to an index word used by the compiler for overlays. It will occur only when the procedure is not in memory.

S — S = 1 Section
0 Paragraph

Usage — Usage of data-name item

1 Display
2 Computational
4 Computational-1
6 Mixed (group)

Class — Class of data-name item

0 Not specified
1 Alphabetic
2 Numeric
3 Alphanumeric
7 Mixed (group)

Type — Octal Value

4x	Group item; x can be any value but 4
44	File
5x	Elementary item; x can be any value
24	Paragraph
26	Section

BCP — Beginning Character Position

0	Starting position is 0
1	Starting position is 6
2	Starting position is 12
.	.
,	.
.	.
9	Starting position is 54

Size — SIZE of a data-name item as defined in the Data Division; computational-1 items may be one or two words.

Synch — Synchronization of a data-name item as defined in the Data Division

0	Not synchronized
1	Synchronized right
2	Synchronized left

Just — Justification of a data-name item as specified in the Data Division

0	Not justified (justified left)
1	Justified right

Sign — Sign of a numeric item

0	Unsigned
1	Signed

Point — Point Location of a numeric item

bit position in field	
1st	0 assumed point
2nd	1 actual point
3rd - 7th	value of positions from right

Unassigned fields do not necessarily contain specific values.

SAMPLE PROGRAMS F

Four sample programs are included in this appendix. The first is a typical COBOL program for updating a master file and producing a report. The other three brief examples illustrate new features in 6400/6500/6600 COBOL: mass storage input-output, sort, and subcompile. A sample of the Report Writer is given in Chapter 5.

EXAMPLE OF COBOL PROGRAM

```
001010 IDENTIFICATION DIVISION.                                          NEW-EX
001020 PROGRAM-ID.  PUBLICATION-EXAMPLE.                                 NEW-EX
001030 AUTHOR.  CONTROL DATA CORPORATION.                                NEW-EX
001040 INSTALLATION.  PALO ALTO.                                         NEW-EX
001050 DATE-WRITTEN.  MARCH-1963.                                        NEW-EX
001060 DATE-COMPILED.   01/03/66  VERSION  2.1 3400P3600 COBOL COMPILER  NEW-EX
001070 SECURITY.  NONE.                                                  NEW-EX
001080 REMARKS.  THIS IS A MASTER UPDATE EXAMPLE WITH A SPECIAL REPORT    NEW-EX
001090     FILE.                                                        NEW-EX
002010 ENVIRONMENT DIVISION.                                             NEW-EX
002020 CONFIGURATION SECTION.                                            NEW-EX
002030 SOURCE-COMPUTER.  6600.                                           NEW-EX
002040 OBJECT-COMPUTER.  6600.                                           NEW-EX
002050 INPUT-OUTPUT SECTION.                                             NEW-EX
002060 FILE-CONTROL.  SELECT MASTER-FILE ASSIGN TO TAPE01 FOR            NEW-EX
002070     MULTIPLE REEL.  SELECT UPDATED-MASTER-FILE RENAMING MASTER-   NEW-EX
002080-    FILE ASSIGN TO TAPE02  FOR MULTIPLE REEL.  SELECT REPORT-     NEW-EX
002090-    FILE ASSIGN TO PTAPE01.     SELECT DETAIL-FILE ASSIGN TO      NEW-EX
002100     INPUT.            SELECT ERROR-FILE ASSIGN TO OUTPUT.        NEW-EX
002110 I-O-CONTROL.  SAME RECORD AREA FOR MASTER-FILE,  UPDATED-MASTER-  NEW-EX
002120-    FILE.                                                        NEW-EX
003010 DATA DIVISION.                                                    NEW-EX
003020 FILE SECTION.                                                     NEW-EX
003030 FD  MASTER-FILE  RECORDING MODE IS DECIMAL LABEL RECORDS ARE      NEW-EX
003040     STANDARD VALUE OF IDENTIFICATION IS ≠INVENT-MASTER =         NEW-EX
003050     DATE-WRITTEN IS DATE-WORD  DATA RECORD IS MASTER.            NEW-EX
003070     01  MASTER SIZE IS 72 ALPHANUMERIC CHARACTERS.               NEW-EX
003080         03  ITEM-NUMBER          PICTURE IS 9(4).               NEW-EX
003090         03  ITEM-NAME  SIZE IS 20 AN DISPLAY CHARACTERS.        NEW-EX
003100             05  GENERAL-CLASS    PICTURE IS X(10).              NEW-EX
003110             05  DETAIL-NAME      PICTURE IS X(10).              NEW-EX
003120         03  ON-HAND-QUAN         PICTURE IS 9(5).               NEW-EX
003130         03  REORDER-LEVEL        PICTURE IS 9(4).               NEW-EX
003140         03  RETAIL-PRICE         PICTURE IS 9(3)V99.            NEW-EX
003150         03  WHOLESALE-LEVEL      PICTURE IS 9(3).               NEW-EX
003160         03  WHOLESALE-PRICE      PICTURE IS 9(3)V99.            NEW-EX
003170         03  YEAR-TO-DATE-SALES   PICTURE IS 9(5).               NEW-EX
003180         03  YEAR-TO-DATE-PROFIT  PICTURE IS 9(6)V99.            NEW-EX
003190         03  LAST-YEAR-SALES      PICTURE IS 9(5).               NEW-EX
003200         03  LAST-YEAR-PROFIT     PICTURE IS 9(6)V99.            NEW-EX
003210 FD  DETAIL-FILE  RECORDING MODE IS DECIMAL LABEL RECORDS ARE      NEW-EX
003220     OMITTED DATA RECORD IS DETAIL.                               NEW-EX
```

```
003230 01  DETAIL.                                                          NEW-EX
003240         03  ITEM-NUMBER          PICTURE IS 9(4).                  NEW-EX
003250         03  ACTION-CODE          PICTURE IS 9.                     NEW-EX
003260             88  SALE VALUE IS 1.                                   NEW-EX
003270             88  RECEIPT VALUE IS 2.                                NEW-EX
003280         03  QUANTITY             PICTURE IS 9(3).                  NEW-EX
003290         03  VENDOR-IDENT SIZE IS 8 NUMERIC DIGITS.                 NEW-EX
003300             06  VENDOR-CLASS     PICTURE IS 9(3).                  NEW-EX
003310             06  VENDOR-NUMBER    PICTURE IS 9(5).                  NEW-EX
003320         03  CUSTOMER-IDENT SIZE IS 8 NUMERIC DIGITS.               NEW-EX
003330             08  CUST-CLASS       PICTURE IS 9(3).                  NEW-EX
003340             08  CUST-NUMBER      PICTURE IS 9(5).                  NEW-EX
003350 FD  REPORT-FILE RECORDING MODE IS DECIMAL LABEL RECORDS ARE        NEW-EX
003360     OMITTED DATA RECORD IS RECORD-LINE.                            NEW-EX
003370      01  RECORD-LINE SIZE IS 120 AN DISPLAY CHARACTERS.            NEW-EX
003380          03  FILLER               PICTURE IS X(5).                NEW-EX
003390          03  ITEM-NUMBER          PICTURE IS 9(4).                NEW-EX
003400          03  FILLER               PICTURE IS X(4).                NEW-EX
003410          03  ITEM-NAME  SIZE IS 20 AN DISPLAY CHARACTERS.         NEW-EX
003420              05  GENERAL-CLASS    PICTURE IS X(10).               NEW-EX
003430              05  DETAIL-NAME      PICTURE IS X(10).               NEW-EX
003440          03  FILLER               PICTURE IS X(4).                NEW-EX
003450          03  YEAR-TO-DATE-SALES   PICTURE IS Z(5).                NEW-EX
003460          03  FILLER               PICTURE IS X(4).                NEW-EX
003470          03  PROJECTOR-SALES      PICTURE IS Z(5).                NEW-EX
003480          03  FILLER               PICTURE IS X(4).                NEW-EX
003490          03  LAST-YEAR-SALES      PICTURE IS Z(5).                NEW-EX
003500          03  FILLER               PICTURE IS X(4).                NEW-EX
003510          03  YEAR-TO-DATE-PROFIT  PICTURE IS $*****9.99.          NEW-EX
003520          03  FILLER               PICTURE IS X(4).                NEW-EX
003530          03  PROJECTED-PROFIT     PICTURE IS $*****9.99.          NEW-EX
003540          03  FILLER               PICTURE IS X(4).                NEW-EX
003550          03  LAST-YEAR-PROFIT     PICTURE IS $*****9.99.          NEW-EX
003560          03  FILLER               PICTURE IS X(4).                NEW-EX
003570          03  SALES-COMP           PICTURE IS X(3).                NEW-EX
003580          03  FILLER               PICTURE IS X(4).                NEW-EX
003590          03  PROFIT-COMP          PICTURE IS X(3).                NEW-EX
003600          03  FILLER               PICTURE IS X(4).                NEW-EX
003610 FD   ERROR-FILE RECORDING MODE IS DECIMAL LABEL RECORDS ARE OMIT   NEW-EX
003620-     TED DATA RECORD IS ERROR-REC.                                 NEW-EX
003630      01  ERROR-REC SIZE IS 120 AN DISPLAY CHARACTERS.              NEW-EX
003631          02  FILLER               PICTURE IS X.                   NEW-EX
003640          02  BAD-REC              PICTURE IS X(72).               NEW-EX
003650          02  MESSAGE              PICTURE IS X(32).               NEW-EX
003651          02  FILLER               PICTURE IS X(15).               NEW-EX
003660 WORKING-STORAGE SECTION.                                          NEW-EX
003670      77  PATTERN-WORD             PICTURE IS X(8).                NEW-EX
003680      77  DAY-OF-WEEK              PICTURE IS X(8).                NEW-EX
003690      77  HEADER-LINE-COUNTER SIZE IS 1 NUMERIC DIGIT VALUE IS 0.  NEW-EX
003700      77  TITLE-LINE-COUNTER  SIZE IS 2 NUMERIC DIGITS VALUE IS 0. NEW-EX
003710      77  NEW-YEAR-IND  SIZE IS 1 NUMERIC DIGIT VALUE IS 0.        NEW-EX
003720      77  WORK-PROJ-SALES          PICTURE IS 9(5).                NEW-EX
003730      77  SALES-DIFF               PICTURE IS 9(5).                NEW-EX
003470      77  PERCENT-SALES-DIFF       PICTURE IS 9(3).                NEW-EX
003750          88  LO  VALUES ARE 0 THRU 10.                            NEW-EX
```

```
003760         88  AVERAGE VALUES ARE 11 THRU 20.                          NEW-EX
003770         88  GOOD VALUES ARE 21 THRU 50.                             NEW-EX
003780         88  VERY-GOOD VALUES ARE 51 THRU 75.                        NEW-EX
003790     77  WORK-PROJ-PROFIT          PICTURE IS 9(6)V99.               NEW-EX
003800     77  PROFIT-DIFF               PICTURE IS 9(6)V99.               NEW-EX
003810     77  PERCENT-PROFIT-DIFF       PICTURE IS $999                   NEW-EX
003820         88  P-LOW VALUES ARE 0 THRU 10.                             NEW-EX
003830         88  P-AVERAGE VALUES ARE 11 THRU 20.                        NEW-EX
003840         88  P-GOOD VALUES ARE 21 THRU 50.                           NEW-EX
003850         88  P-VERY-GOOD VALUES ARE 51 THRU 75.                      NEW-EX
003860     77  MASTER-END-IND SIZE 1 NUMERIC COMPUTATIONAL DIGIT VALUE     NEW-EX
003861         IS 0.                                                        NEW-EX
003862     77  RESULT                    PICTURE IS 9(6)V99.               NEW-EX
003863     77  RESULT-A                  PICTURE IS 9(6)V99.               NEW-EX
003864     77  INT-RESULT                PICTURE IS 9(8).                  NEW-EX
003865     77  INT-RESULT-A              PICTURE IS 9(3).                  NEW-EX
003866     77  WORK-RESULT               PICTURE IS 9(8).                  NEW-EX
003867     77  INTER-RESULT              PICTURE IS 9(8).                  NEW-EX
003868     77  INTER-RESULT-A            PICTURE IS 9(3).                  NEW-EX
003869     77  WORKING-RESULT            PICTURE IS 9(10).                 NEW-EX
003870     01  DATE-WORD  SIZE IS 8 AN CHARACTERS.                         NEW-EX
003880         02  MONTH  SIZE IS 2 NUMERIC COMPUTATIONAL DIGITS.          NEW-EX
003890         02  FILLER SIZE IS 1 AN CHARACTER.                          NEW-EX
003900         02  DAY    SIZE IS 2 NUMERIC COMPUTATIONAL DIGITS.          NEW-EX
003910         02  FILLER SIZE IS 1 AN CHARACTER.                          NEW-EX
003920         02  YEAR   SIZE IS 2 NUMERIC COMPUTATIONAL DIGITS.          NEW-EX
003930     01  HEADER-LINE SIZE IS 120 AN CHARACTERS.                      NEW-EX
003940         02 HEADER SIZE IS 112 AN CHARACTERS VALUE IS ≠ 1           NEW-EX
003950-        ≠                    S A L E S   A N D   P R O F I T   I N NEW-EX
003960-        ≠ D I C A T O R                                            NEW-EX
003970         02  CURRENT-DATE COPY DATE-WORD.                            NEW-EX
003980     01  TITLE-LINE SIZE IS 120 AN CHARACTERS.                       NEW-EX
003990         02  PRINTING-CONTROL SIZE IS 1 NUMERIC CHARACTER VALUE      NEW-EX
004000         IS 0.                                                        NEW-EX
004010         02  TITLE SIZE IS 119 AN CHARACTERS VALUE IS ≠       ITE   NEW-EX
004020-        ≠M  DESCRIPTION       YTD-SAL  PROJSAL  LASTSAL  YTD-PR    NEW-EX
004030-        ≠OFIT    PROJPROFIT    LAST-PROFIT   S-COMP   PCOMP ≠.    NEW-EX


004040 PROCEDURE DIVISION.                                                NEW-EX
004050 START. ACCEPT DATE-WORD.  OPEN INPUT                               NEW-EX
004060     MASTER-FILE DETAIL-FILE.  ACCEPT CURRENT-DATE.  ACCEPT         NEW-EX
004070     DAY-OF-WEEK.  IF YEAR OF CURRENT-DATE IS GREATER THAN YEAR     NEW-EX
004080     OF DATE-WORD MOVE 1 TO NEW-YEAR-IND ELSE NEXT SENTENCE.        NEW-EX
004090     MOVE CURRENT-DATE TO DATE-WORD.  MOVE DAY-OF-WEEK TO PATTERN-  NEW-EX
004100-    WORD.  OPEN OUTPUT UPDATED-MASTER-FILE REPORT-FILE ERROR.      NEW-EX
004110-    FILE. READ MASTER-FILE AT END GO TO END-MASTER. READ DETAIL-   NEW-EX
004115-    FILE AT END GO TO END-DETAIL.  IF NEW-YEAR-IND IS              NEW-EX
004120     EQUAL TO 1 GO TO NEW-YEAR-PROC ELSE GO TO TEST.                NEW-EX
004130 TEST.  IF ITEM-NUMBER OF MASTER-FILE IS EQUAL TO ITEM-NUMBER OF    NEW-EX
004140     DETAIL GO TO UPDATE. IF ITEM-NUMBER OF MASTER-FILE LESS THAN   NEW-EX
004150     ITEM-NUMBER OF DETAIL GO TO WRITEOUT ELSE GO TO BADSEQ.        NEW-EX
004160 UPDATE.  IF SALE NEXT SENTENCE ELSE GO TO RECEIVED.  SUBTRACT      NEW-EX
004170     QUANTITY FROM ON-HAND-QUAN OF MASTER-FILE.  IF ON-HAND-QUAN    NEW-EX
004180     OF MASTER-FILE IS LESS-EQUAL TO REORDER-LEVEL OF MASTER-FILE   NEW-EX
004190     PERFORM REORDER ELSE NEXT SENTENCE. ADD QUANTITY TO YEAR-TO-D  NEW-EX
```

```
004200-     ATE-SALES OF MASTER-FILE. IF QUANTITY IS GREATER-EQUAL WHOLES   NEW-EX
004210-     ALE-LEVEL OF MASTER-FILE GO TO WHOLESALE ELSE GO TO RETAIL.     NEW-EX
004220 RECEIVED.  IF NOT RECEIPT GO TO BADCODE ELSE NEXT SENTENCE.  ADD      NEW-EX
004230      QUANTITY TO ON-HAND-QUAN OF MASTER-FILE.  GO TO RETURN.         NEW-EX
004240 RETAIL, MULTIPLY QUANTITY BY RETAIL-PRICE OF MASTER-FILE GIVING       NEW-EX
004250      RESULT. MULTIPLY RESULT BY .07 GIVING RESULT-A ROUNDED. ADD     NEW-EX
004260      RESULT-A TO YEAR-TO-DATE-PROFIT OF MASTER-FILE ROUNDED.         NEW-EX
004265      GO TO RETURN.                                                   NEW-EX
004270 WHOLESALE. MULTIPLY QUANTITY BY WHOLESALE-PRICE OF MASTER-FILE GI     NEW-EX
004280-     VING RESULT. MULTIPLY RESULT BY .05 GIVING RESULT-A ROUNDED.    NEW-EX
004290      ADD RESULT-A TO YEAR-TO-DATE-PROFIT OF MASTER-FILE. GO TO       NEW-EX
004295      RETURN.                                                         NEW-EX
004300 BADCODE.  MOVE DETAIL TO BAD-REC.  MOVE ≠ BAD ACTION CODE ≠ TO       NEW-EX
004310      MESSAGE.  WRITE ERROR-REC.  GO TO RETURN.                       NEW-EX
004320 BADSEQ, MOVE DETAIL TO BAD-REC  MOVE ≠ NO MASTER REC FOR DETAIL R     NEW-EX
004330-     ≠EC≠ TO MESSAGE, WRITE ERROR-REC. GO TO RETURN.                 NEW-EX
004340 REORDER. MOVE MASTER OF MASTER-FILE TO BAD-REC. MOVE ≠REORDER≠ TO     NEW-EX
004350      MESSAGE.  WRITE ERROR-REC.
004360 NEW-YEAR-PROC. MOVE YEAR-TO-DATE-SALES OF MASTER-FILE TO LAST-YEA     NEW-EX
004370-     R-SALES OF UPDATED-MASTER-FILE. MOVE YEAR-TO-DATE-PROFIT OF     NEW-EX
004380      MASTER-FILE TO LAST-YEAR-PROFIT OF UPDATED-MASTER-FILE. MOVE    NEW-EX
004390      ZEROS TO YEAR-TO-DATE-SALES OF UPDATED-MASTER-FILE. MOVE ZERO   NEW-EX
004410-     S TO YEAR-TO-DATE-PROFIT OF UPDATED-MASTER-FILE. GO TO TEST.    NEW-EX
004430 WRITEOUT.  IF HEADER-LINE-COUNTER IS NOT EQUAL TO 0                   NEW-EX
004440      MOVE 1 TO PRINTING-CONTROL THEN IF                              NEW-EX
004450      TITLE-LINE-COUNTER IS NOT EQUAL TO 30 GO TO ANALYSIS-PROC       NEW-EX
004460      ELSE MOVE ZEROS TO TITLE-LINE-COUNTER GO TO TITLES-PROC ELSE    NEW-EX
004470      GO TO HEADER-PROC.                                              NEW-EX
004480 HEADER-PROC. WRITE RECORD-LINE FROM HEADER-LINE. ADD 1 TO HEADER-     NEW-EX
004490-     LINE-COUNTER.                                                   NEW-EX
004500 TITLES-PROC.  WRITE RECORD-LINE FROM TITLE-LINE.                      NEW-EX
004510 ANALYSIS-PROC.  MOVE SPACES TO RECORD-LINE.  MOVE CORRESPONDING       NEW-EX
004520      MASTER OF MASTER-FILE TO RECORD-LINE. MULTIPLY YEAR-TO-DATE-S   NEW-EX
004530-     ALES OF MASTER-FILE BY 360 GIVING INT-RESULT. MULTIPLY MONTH    NEW-EX
004540      OF CURRENT-DATE BY 30 GIVING INT-RESULT-A. ADD DAY OF CURRENT   NEW-EX
004550-     =DATE. -30 TO INT-RESULT-A. DIVIDE INT-RESULT-A INTO INT-RESU   NEW-EX
004560-     LT GIVING WORK-PROJ-SALES. SUBTRACT LAST-YEAR-SALES OF MASTER   NEW-EX
004570-     =FILE FROM WORK-PROJ-SALES GIVING SALES-DIFF. MULTIPLY SALES-   NEW-EX
004580-     DIFF BY 100 GIVING WORK-RESULT. DIVIDE LAST-YEAR-SALES OF       NEW-EX
004590      MASTER-FILE INTO WORK-RESULT GIVING PERCENT-SALES-DIFF ROUNDE   NEW-EX
004600-     D. IF PERCENT-SALES-DIFF IS LESS 0 MOVE ≠DEC≠ TO SALES-COMP.    NEW-EX
004610      IF LO  MOVE ≠LOW≠ TO SALES-COMP. IF AVERAGE MOVE ≠AVG≠ TO       NEW-EX
004611      SALES-COMP.                                                     NEW-EX
004620      IF GOOD MOVE ≠GD.≠ TO SALES-COMP. IF VERY-GOOD MOVE ≠VGD≠ TO    NEW-EX
004630      SALES-COMP ELSE MOVE ≠EXC≠ TO SALES-COMP.  MULTIPLY YEAR-TO-    NEW-EX
004640-     DATE-PROFIT OF MASTER-FILE BY 360 GIVING INTER-RESULT. MULTIP   NEW-EX
004650-     LY MONTH OF CURRENT-DATE BY 30 GIVING INTER-RESULT-A. ADD DAY   NEW-EX
004660      OF CURRENT-DATE. -30 TO INTER-RESULT-A. DIVIDE INTER-RESULT-A   NEW-EX
004670      INTO INT-RESULT GIVING WORK-PROJ-PROFIT. SUBTRACT LAST-YEAR-    NEW-EX
004680-     PROFIT OF MASTER-FILE FROM WORK-PROJ-PROFIT GIVING PROFIT-      NEW-EX
004690-     DIFF.  MULTIPLY PROFIT-DIFF BY 100 GIVING WORKING-RESULT        NEW-EX
004700      DIVIDE LAST-YEAR-PROFIT OF MASTER-FILE INTO WORKING-RESULT      NEW-EX
004710      GIVING PERCENT-PROFIT-DIFF ROUNDED. IF PERCENT-PROFIT-DIFF IS   NEW-EX
004720      NEGATIVE MOVE ≠DEC≠ TO PROFIT-COMP.  IF P-LOW MOVE ≠LOW≠ TO     NEW-EX
004730      PROFIT-COMP.  IF P-AVERAGE MOVE ≠AVG≠ TO PROFIT-COMP.  IF       NEW-EX
```

```
004740      P-GOOD MOVE ≠GD ≠ TO PROFIT-COMP.  IF P-VERY-GOOD MOVE ≠VGD≠    NEW-EX
004750      TO PROFIT-COMP ELSE MOVE ≠EXC≠ TO PROFIT-COMP. MOVE WORK-PROJ   NEW-EX
004760      -SALES TO PROJECTED-SALES.  MOVE WORK-PROJ-PROFIT TO            NEW-EX
004770      PROJECTED-PROFIT.  GO TO WRITM.                                 NEW-EX
004780 WRITM. WRITE RECORD-LINE. ADD 1 TO TITLE-LINE-COUNTER.  WRITE MAS    NEW-EX
004790-     TER OF UPDATED-MASTER-FILE. READ MASTER-FILE AT END MOVE 1 TO   NEW-EX
004800      MASTER-END-IND GO TO END-MASTER.  IF NEW-YEAR-IND IS EQUAL TO   NEW-EX
004810      1 GO TO NEW-YEAR-PROC ELSE GO TO TEST.                          NEW-EX
004820 RETURN.  READ DETAIL-FILE AT END GO TO END-DETAIL.  GO TO TEST.      NEW-EX
004830 END-DETAIL.  IF MASTER-END-IND IS EQUAL TO 1 GO TO CLOSE-FILES       NEW-EX
004840      ELSE MOVE 9999 TO ITEM-NUMBER OF DETAIL GO TO WRITEOUT.         NEW-EX
004850 END-MASTER.  READ DETAIL-FILE INTO BAD-REC AT END GO TO END.         NEW-EX
004860-     DETAIL.  MOVE ≠NO MASTER REC FOR DETAIL REC≠ TO MESSAGE.        NEW-EX
004870      WRITE ERROR-REC.  GO TO END-MASTER.                             NEW-EX
004880 CLOSE-FILES.  CLOSE MASTER-FILE DETAIL-FILE ERROR-FILE UPDATED-      NEW-EX
004890-     MASTER-FILE REPORT-FILE.  STOP RUN.                             NEW-EX
```

EXAMPLE OF SORT

The program SORT-1 establishes a file which program SORT-2 uses. The same file is sorted on two different keys in two different programs.

```
RMM, 100, 75000, 17.
REQUEST, TAPE01, MT.
REQUEST, TAPE02, MT.
COBOL.
LGO.
```

(7,8,9 card)

```
        IDENTIFICATION DIVISION.
        PROGRAM-ID. SORT-1.
        AUTHOR.  CONTROL DATA SORT EXAMPLE PART 1.
        DATE-COMPILED.
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER. 6400
        OBJECT-COMPUTER. 6400
        SPECIAL-NAMES.
            OUTPUT IS PRINT.
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
            SELECT GEN-FILE ASSIGN TO TAPE01.
            SELECT SORT-FILE ASSIGN TO TAPE02.
        DATA DIVISION.
        FILE SECTION.
        FD  GEN-FILE
                BLOCK CONTAINS 10 RECORDS.
                LABEL RECORDS OMITTED.
                DATA RECORD IS GEN-REC.
            01  GEN-REC.
                02 IDENT-1  PICTURE 9(8).
                02 FILL-1   PICTURE X(20).
                02 IDENT-2.
                   03  ID-2A PICTURE 99.
                   03  ID-2B PICTURE XXX.
                02 FILL-2   PICTURE X(60).
                02 IDENT-3  PICTURE X(6).
        SD  SORT-FILE
                DATA RECORD IS SORT-REC.
            01  SORT-REC COPY GEN-REC.
        WORKING-STORAGE SECTION.
            77  A   PICTURE 9(8) VALUE 00250000.
            77  B   PICTURE 9(2) VALUE 00.
            77  C   PICTURE XXX  VALUE SPACES.
            77  N   PICTURE 9 VALUE 0.
            77  X   PICTURE 9(8) VALUE ZEROS.
            77  Y   PICTURE X(5) VALUE SPACES.
            77  Z   PICTURE X(6) VALUE SPACES.
            77  RC   PICTURE 9999 VALUE ZERO.
            01  TABLPHB  PICTURE X(24) VALUE  ≠ABCDEFGHIJKLMNOPQRSTUVWX≠.
            01  TABLPHA REDEFINES TABLPHB.
                02 TA  PICTURE X(6) OCCURS 4.
```

```
PROCEDURE DIVISION.
START SECTION.
N1. NOTE THIS SECTION WILL GENERATE A FILE TO BE USED BY SORT-1
    AND SORT-2.
ST. DISPLAY ≠1    BEGIN SORT-1≠ UPON PRINT.
    OPEN OUTPUT GEN-FILE.
    MOVE ALL ≠GEN-FILE≠ TO FILL-1, IN GEN-REC.
    MOVE ALL ≠MORE-GEN-FILE≠ TO FILL-2, IN GEN-REC.
    PERFORM ST1 THRU ST3 250  TIMES. GO TO ST4.
ST1.
     PERFORM ST2 VARYING N FROM 1 BY 1 UNTIL N GR 4.
    GO TO ST3.
ST2.
    MOVE TA (N) TO IDENT-3 OF GEN-REC.
    MOVE TA (N) TO ID-2B OF GEN-REC.
    ADD 1 TO B. IF B = 99 MOVE ZERO TO B.
    MOVE B TO ID-2A, IN GEN-REC.
    SUBTRACT 5 FROM A.
    MOVE A TO IDENT-1, IN GEN-REC.
    WRITE GEN-REC.
ST3. EXIT
ST4. MOVE 255000 TO A.
    PERFORM ST1 THRU ST3 250 TIMES.
    CLOSE GEN-FILE.
    SORT SORT-FILE ON ASCENDING KEY,
        IDENT-1 OF SORT-REC, IDENT-2 OF SORT-REC, IDENT-3 OF
        SORT-REC,
        INPUT PROCEDURE IS IN1 THRU IN2,
        OUTPUT PROCEDURE IS OUT1.
IN1 SECTION.
IMP. OPEN INPUT GEN-FILE.
I1. READ GEN-FILE AT END GO TO IN2P.
    MOVE CORRESPONDING GEN-REC TO SORT-REC.
    MOVE ALL ≠SORT-FILE≠ TO FILL-1 OF SORT-REC.
    RELEASE SORT-REC.
    GO TO I1.
IN2 SECTION.
IN2P.
    CLOSE GEN-FILE.
OUT1 SECTION.
OT1.
    RETURN SORT-FILE, AT END GO TO ENDIT1.
    ADD1 TO RC.
    IF IDENT-1 OF SORT-REC GQ X AND IDENT-2 OF SORT-REC GQ Y
    AND IDENT-3 OF SORT-REC GQ Z PERFORM PASS ELSE PERFORM FAIL.
    GO TO OT1.
ENDIT SECTION.
ENDIT1.
    IF RC = 2000 GO TO ENDIT2.
    DISPLAY ≠0*** FAILURE ON RECORD COUNT, RC SHOULD = 2000, RC =
-   ≠ ≠ RC UPON PRINT.
ENDIT2.
    DISPLAY ≠0    END SORT-1= IF NO FAIL MESSAGES TEST OK≠ UPON
    PRINT.
```

```
    STOP RUN.
P-F SECTION.
PASS.
    MOVE IDENT-1 OF SORT-REC TO X.
    MOVE IDENT-2 OF SORT-REC TO Y.
    MOVE IDENT-3 OF SORT-REC TO Z.
FAIL.
    DISPLAY ≠0*** FAILURE ON SEQUENCE CHECK OF SORTED FILE.≠
    UPON PRINT.
    DISPLAY ≠0    IDENT-1 = ≠ IDENT-1 OF SORT-REC ≠ X = ≠  X
    UPON PRINT.
    DISPLAY ≠0    IDENT-2 = ≠ IDENT-2 OF SORT-REC ≠  Y - ≠ Y
    UPON PRINT.
    DISPLAY ≠0    IDENT-3 = ≠ IDENT-3 OF SORT-REC ≠ Z = ≠ Z
    UPON PRINT.
```

(6,7,8,9 card)

(7,8,9 card)

```
IDENTIFICATION DIVISION
PROGRAM-ID. SORT-2.
AUTHOR. CONTROL DATA SORT EXAMPLE PART 2.
DATE-WRITTEN.  5 JANUARY 1967.
DATE-COMPILED.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. 6400.
OBJECT-COMPUTER. 6400.
SPECIAL-NAMES.
    OUTPUT IS PRINT.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT SORT-FILE-1 ASSIGN TO TAPE01.
    SELECT FILE-2 ASSIGN TO TAPE02.
    SELECT FILE-3 ASSIGN TO TAPE03.
    SELECT SORT-FILE-2 ASSIGN TO TAPE04.
DATA DIVISION.
FILE SECTION.
FD  FILE-2  BLOCK CONTAINS 10 RECORDS
        LABEL RECORDS ARE OMITTED
        DATA RECORD IS TWOFILE.
01  TWOFILE.
    02  ID-1     PICTURE 9(8).
    02  FILL-1   PICTURE X(20).
    02  ID-2     PICTURE X(5).
        03  ID-2A   PICTURE 99.
        03  ID-2B   PICTURE XXX.
    02  FILL-2   PICTURE X(60).
    02  ID-3     PICTURE X(6).
FD  FILE-3 BLOCK CONTAINS 20 RECORDS
        LABEL RECORDS ARE OMITTED
        DATA RECORD IS TREFILE.
```

```
01  TREFILE COPY TWOFILE.
SD  SORT-FILE-1
    DATA RECORD IS ONESORT.
01  ONESORT COPY TWOFILE.
SD  SORT-FILE-2
    DATA RECORD IS TWOSORT.
01  TWOSORT COPY TWOFILE.
WORKING-STORAGE SECTION.
    77  I-1    PICTURE IS 9(8)    VALUE IS 0.
    77  I-2    PICTURE IS X(5)    VALUE IS BLANK.
    77  I-3    PICTURE IS X(6)    VALUE IS ZZZZZZ.
    77  COUNTER  PICTURE IS 9(4) VALUE IS 0.
PROCEDURE DIVISION.
START SECTION.
ST. DISPLAY ≠1   BEGIN TEST ≠ UPON PRINT.
    SORT SORT-FILE-1 ON ASCENDING KEY I-1 I-2 ON DESCENDING
    KEY I-3 USING FILE-2 GIVING FILE-3.
VERIFICATION-1.
    OPEN INPUT FILE-3.
    READ FILE-3 AT END GO TO END-1.
    IF ID-1 is GQ I-1 AND
       ID-2 IS GQ I-2 AND
       ID-3 IS LQ I-3 PERFORM PASS ELSE PERFORM FAIL.
PASS.
    MOVE ID-1 TO I-1. MOVE ID-2 TO I-2. MOVE ID-3.
    ADD 1 TO COUNTER.
FAIL.
    DISPLAY ≠TEST OF SORT FAILURE≠ UPON PRINT.
    DISPLAY ≠I-1 EQUALS ≠ I-1      UPON PRINT.
    DISPLAY ≠I-2 EQUALS ≠ I-2      UPON PRINT.
    DISPLAY ≠I-3 EQUALS ≠ I-3      UPON PRINT.
    DISPLAY ≠COUNT EQUALS ≠ COUNTER UPON PRINT.
END-1.
    IF COUNTER EQUALS 2000 GO TO END-2 ELSE NEXT SENTENCE.
    DISPLAY ≠RECORD COUNT FAILURE
-           ≠COUNTER SHOULD BE 2000
-           ≠COUNTER IS ≠ COUNTER UPON PRINT.
    GO TO END-2.
END-2.  CLOSE FILE-3. GO TO BEGIN SECTION.
BEGIN SECTION.
    SORT SORT-FILE-2 ON ASCENDING KEY I-1 I-2  ON DESCENDING
    KEY I-3 INPUT PROCEDURE IS STEP-1 SECTION
    OUTPUT PROCEDURE IS STEP-2 SECTION.
STEP-1 SECTION.
    MOVE TWOFILE TO TWOSORT.
    RELEASE TWOSORT.
STEP-2 SECTION.
    RETURN SORT-FILE-2 AT END GO TO AT END.
        IF ID-1 IS GQ I-1 AND
           ID-2 IS GQ I-2 AND
           ID-3 IS LQ I-3 PERFORM PASS ELSE PERFORM FAIL.
    AT END.  IF COUNTER EQUALS 2000 DISPLAY ≠END OF TEST≠ ELSE
    DISPLAY ≠RECORD COUNT FAILURE
-           ≠ COUNTER SHOULD BE 2000
```

```
-           ≠COUNTER IS ≠ COUNTER UPON PRINT.
      STOPIT.
          STOP RUN.
```

(7,8,9 card)

EXAMPLE USING MASS STORAGE I/O

```
 RMM,100,75000,17.
 REQUEST,DISK-1,D0100.
 REQUEST,DISK-2,D0100.
 COBOL.
 LGO.
(7,8,9 card)
        IDENTIFICATION DIVISION.
        PROGRAM-ID.   MASS-TWO.
        AUTHOR. CONTROL DATA MASS STORAGE EXAMPLE.
        DATE-WRITTEN.  12 JANUARY 1967.
        DATE-COMPILED.
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER.  6400.
        OBJECT-COMPUTER.  6400.
        SPECIAL-NAMES.
            OUTPUT IS PRINT.  INPUT IS PUT-IN.
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
            SELECT TWIGGS ASSIGN TO DISK01
            FILE LIMIT IS 2000
            ACCESS MODE IS RANDOM
            PROCESSING MODE IS SEQUENTIAL
            SYMBOLIC KEY IS SYM-KEY-1.
            SELECT HOHUM ASSIGN TO DISK02
            FILE LIMIT IS 100
            ACCESS MODE IS RANDOM
            SYMBOLIC KEY IS SYM-KEY-2.
        DATA DIVISION.
        FILE SECTION.
        FD  TWIGGS
            LABEL RECORDS ARE OMITTED
            DATA RECORD IS ROOTS.
            01  ROOTS.
                02 KEY-1      PICTURE X(7).
                02 GARBAGE-1 PICTURE X(113).
        FD  HOHUM
            LABEL RECORDS ARE OMITTED
            DATA RECORD IS YAWN.
            01  YAWN.
                02 KEY-2      PICTURE X(5).
                02 GARBAGE-2 PICTURE
```

```
WORKING-STORAGE.
    77  SYM-KEY-1     PICTURE X(7).
    77  SYM-KEY-2     PICTURE X(5).
    77  N             PICTURE 9999.
    77  COUNT         PICTURE 9999  VALUE IS 0.
    01  TABLE.
        02  TABLE-A PICTURE X(25) VALUE IS ≠ABCDEFGHIJKLMNOPQRST
-       ≠UVWXY≠.
        02  TABLE-B REDEFINES TABLE-A PICTURE X(5) OCCURS 5
        TIMES.
    01  CARD-IN-1.
        02  CARD-KEY-1  PICTURE X(7).
        02  FILLER      PICTURE X(113).
    01  CARD-IN-2.
        02  CARD-KEY-2  PICTURE X(5).
        02  FILLER      PICTURE X(115).
PROCEDURE DIVISION.
START.
    OPEN INPUT-OUTPUT TWIGGS, HOHUM.
    PERFORM ABC 1900 TIMES,  GO TO NEXT-STEP.
ABC.
    MOVE ≠COUNTER≠ TO KEY-1.
    MOVE ALL ≠FILE-1≠ TO GARBAGE-1.
    ADD 1 TO ≠COUNTER≠.
    MOVE KEY-1 TO SYM-KEY-1.
    WRITE ROOTS INVALID KEY PERFORM ERROR-OP.
NEXT-STEP.
    PERFORM XYZ VARYING N FROM 1 BY 1 UNTIL N = 6. GO TO FINISH.
XYZ.
    MOVE TABLE-B(N) TO KEY-2.
    MOVE ALL ≠FILE-2≠ TO GARBAGE-2.
    MOVE KEY-2 TO SYM-KEY-2.
    WRITE YAWN INVALID KEY PERFORM ERROR-OP.
FINISH.
    CLOSE     TWIGGS.
BEGIN.
    OPEN I-O TWIGGS.
READ-CARDS-1.
    ACCEPT CARD-IN-1 FROM PUT-IN.
    MOVE CARD-KEY-1 TO SYM-KEY-1.
    READ TWIGGS INVALID KEY PERFORM ERROR-OP.
    IF CARD-KEY-1 = KEY-1
    AND GARBAGE-1 = ALL ≠FILE-1≠ THEN PERFORM PASS-1
    ELSE PERFORM FAIL-1.
    ADD 1 TO COUNT.
    IF COUNT IS LESS THAN 20 GO TO READ-CARDS-1.
CLEAN-UP.
    DISPLAY ≠COUNT SHOULD BE 20   COUNT IS ≠ COUNT UPON PRINT.
    MOVE 0 TO COUNT.
READ-CARDS-2.
    ACCEPT CARD-IN-2 FROM PUT-IN.
    MOVE CARD-KEY-2 TO SYM-KEY-2.
    READ HOHUM INVALID KEY PERFORM ERROR-OP.
    IF CARD-KEY-2 = KEY-2
```

```
           AND GARBAGE-2 = ALL ≠FILE-2≠ THEN PERFORM PASS-1
           ELSE PERFORM FAIL-2.
           ADD 1 TO COUNT.
           IF COUNT IS LESS THAN 5 GO TO READ-CARDS-2.
       CLEAN-UP-2.
           DISPLAY ≠COUNT SHOULD BE 5   COUNT IS ≠ COUNT UPON PRINT.
           MOVE 0 TO COUNT.
       STOPIT.
           CLOSE      TWIGGS, HOHUM.
           STOP RUN.
       ERROR-OP.
           DISPLAY ≠INVALID KEY ERROR.≠.
       PASS-1.
           DISPLAY ≠TEST OK≠ UPON PRINT.
       FAIL-1.
           DISPLAY ≠MASS STORAGE FAILURE≠ UPON PRINT.
           DISPLAY ≠CARD-KEY-1 SHOULS EQUAL KEY-1≠ UPON PRINT.
           DISPLAY ≠CARD-KEY-1 IS ≠ CARD-KEY-1 UPON PRINT.
           DISPLAY ≠KEY-1 IS ≠ KEY-1 UPON PRINT.
           DISPLAY ≠GARBAGE-1 SHOULD BE ALL FILE-1, GARBAGE-1 IS ≠
               GARBAGE-1 UPON PRINT.
       FAIL-2
           DISPLAY ≠MASS STORAGE FAILURE≠ UPON PRINT.
           DISPLAY ≠CARD-KEY-2 SHOULD EQUAL KEY-2≠ UPON PRINT.
           DISPLAY ≠CARD-KEY-2 IS ≠ CARD-KEY-2 UPON PRINT.
           DISPLAY ≠KEY-2 IS ≠ KEY-2 UPON PRINT.
           DISPLAY ≠GARBAGE-2 SHOULD BE ALL FILE-2, GARBAGE IS ≠
               GARBAGE-2 UPON PRINT.
```

(7,8,9 card)

```
0000327
0001854
0001492
0001066
0001776
0001642
0000812
0001010
0000001
0000020
0000893
0000412
0001111
0001234
0000808
0000307
0000064
0001550
0001883
0001396
FGHIJ
ABCDE
PQRST
KLMNO
UVWXY
```

These are data cards for mass storage test program.

(6,7,8,9 card)

EXAMPLE OF SUBCOMPILE CAPABILITIES

```
 RMM,T100, CH75000, P17.
 COBOL.
 COBOL.
 COBOL.
 LGO.
(7,8,9 card)
        IDENTIFICATION DIVISION.
        PROGRAM-ID. BSUBCOMPILE.
        AUTHOR. CONTROL DATA SUBCOMPILE EXAMPLE.
        DATE-WRITTEN.  31 JANUARY 1967.
        DATE-COMPILED.
        REMARKS.
            THIS IS PROGRAM-B WHICH UTILIZES PARA-1 OF ASUBCOMPILE.
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER.  6400.
        OBJECT-COMPUTER.  6400.
        SPECIAL NAMES.
            OUTPUT IS PRINT.
        DATA DIVISION.
        COMMON STORAGE.
            77  A    PICTURE IS 9(3) VALUE IS 1.
            77  B    PICTURE IS 9(3) VALUE IS 2.
            77  C    PICTURE IS 9(3) VALUE IS 0.
        PROCEDURE DIVISION.
        DECLARATIVES.
        BSUBCOM SECTION.
        INSERT-POINT.
            ENTRY PNB-1.
        END DECLARATIVES.
        BEGINNING SECTION.
        PNB-1.
            DISPLAY ≠TEST OF BSUBCOMPILE PROGRAM,≠ UPON PRINT.
            MOVE 2 TO B.
            MOVE 0 TO C.
            PERFORM PARA-1.
            IF C IS NOT EQUAL TO 147 DISPLAY ≠ERROR IN ARITH OR
      -     ≠SUBCOMP FEATURE  -  C SHOULD EQUAL 147, C IS ≠ C UPON PRINT.
(7,8,9 card)
        IDENTIFICATION DIVISION.
        PROGRAM-ID.  CSUBCOMPILE.
        AUTHOR. CONTROL DATA SUBCOMPILE SAMPLE.
        DATE-WRITTEN.  31 JANUARY 1967.
        DATE-COMPILED.
        REMARKS.
            THIS IS PROGRAM-C WHICH UTILIZES PARA-2 OF ASUBCOMPILE.
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER.  6400.
        OBJECT-COMPUTER.  6400.
        SPECIAL NAMES.
            OUTPUT IS PRINT.
```

```
        DATA DIVISION.
        COMMON STORAGE.
            77  A    PICTURE IS 9(3) VALUE IS 1.
            77  B    PICTURE IS 9(3) VALUE IS 2.
            77  C    PICTURE IS 9(3) VALUE IS 0.
        PROCEDURE DIVISION.
        DECLARATIVES.
        CSUBCOM SECTION.
        INSERTION.
            ENTRY PNC-1.
        END DECLARATIVES.
        FURST SECTION.
        PNC-1.
            DISPLAY ≠TEST OF CSUBCOMPILE PROGRAM.≠ UPON PRINT.
            MOVE 147 TO C.
            PERFORM PARA-2.
            IF C IS NOT EQUAL TO 50 DISPLAY ≠ERROR IN ARITH OR SUBCOMP
        -   ≠FEATURE - C SHOULD EQUAL 50, C IS ≠ UPON PRINT.
            GO TO PNC-1.
(7,8,9 card)
        IDENTIFICATION DIVISION.
        PROGRAM-ID. ASUBCOMPILE.
        AUTHOR. CONTROL DATA SUBCOMPILE SAMPLE.
        DATE-WRITTEN.  31 JANUARY 1967.
        DATE-COMPILED.
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER.  6400.
        OBJECT-COMPUTER.  6400.
        SPECIAL NAMES.
            OUTPUT IS PRINT.
        DATA DIVISION.
        COMMON STORAGE.
            77  A    PICTURE IS 9(3) VALUE IS 1.
            77  B    PICTURE IS 9(3) VALUE IS 2.
            77  C    PICTURE IS 9(3) VALUE IS 0.
        PROCEDURE DIVISION.
        DECLARATIVES.
        SUBCOMP SECTION.
        ENTRY-POINTS.
            ENTRY PARA-1, PARA-2.
        END DECLARATIVES.
        START SECTION.
        BEGIN.
            DISPLAY ≠1   BEGIN TEST OF SUBCOMPILE.≠ UPON PRINT.
            PERFORM PARA-1.  GO TO CONTINUE SECTION.
        PARA-1.
            ADD A TO B GIVING C.
            MULTIPLY C BY 49.
        CONTINUE SECTION.
        INITIAL.
            IF C IS NOT EQUAL TO 147 DISPLAY ≠ARITHMETIC ERROR - C SHOULD
        -   ≠EQUAL 147, C IS ≠ C UPON PRINT.
            PERFORM PARA-2,  GO TO ENDING SECTION.
```

```
PARA-2.
    SUBTRACT 47 FROM C.
    DIVIDE C BY 2.
ENDING SECTION.
    IF C IS NOT EQUAL TO 50 DISPLAY ≠ARITHMETIC ERROR - C SHOULD
-   ≠EQUAL 50, C IS ≠ C UPON PRINT.
    PERFORM PNB-1.
    ENTER PNC-1.
    DISPLAY ≠END OF TEST,≠ UPON PRINT.

    STOP RUN.
```

(6,7,8,9 card)

COBOL DIAGNOSTICS G

Compiler diagnostics are printed on the source program listing. Generally, only the catastrophic diagnostics are listed; those that will cause program termination. However, the user may request a listing of extended diagnostics through the LX option on the COBOL control card. A series of asterisks and a diagnostic number are placed before or after the line in error and just to the left of the line number column in the listing. The number is followed by a diagnostic comment. If cross reference line numbers are requested by the LR option, they are printed just to the right of the diagnostic comment. A cross reference indicates the line number in which the referenced item appears.

The compiler recognizes only errors in COBOL syntax; faulty programming logic is not recognized unless it produces a syntax error.

If sequence numbers are specified in columns 1-6, a diagnostic is issued each time the sequence does not ascend. If columns 1-6 are blank, no sequence check is made.

Execution time diagnostics are also produced by the COBOL system. These are displayed on the console when the error is detected. Such errors may or may not cause program termination.

COMPILER DIAGNOSTIC MESSAGES

The types of diagnostic messages produced during compilation are listed below. These types are indicated by a code letter associated with the message.

Code	Meaning
T	Trivial; syntax error, program continues
E	Error; syntax error, program continues
U	Unconventional; syntax error, program continues
C	Catastrophic; program execution terminates

All diagnostic messages produced by the system are separated into functional areas according to the portion of the source program or phase of compilation. The diagnostic numbers fall serially within four arbitrary ranges associated with these functional areas.

Range	Area
1-99	SCAN2
100-129	Identification & Environment Divisions
130-199	General
200-499	Data Division
500-799	Procedure Division (Pass1)
800-999	Procedure Division (Pass2)

Diagnostic messages with corresponding number and type are listed in order in the following pages.

No.	Type	Diagnostic	Discussion
001	T	INCORRECT STARTING COLUMN BEFORE COLUMN 12	Line begins between A and B margins.
002	T	INCORRECT PUNCTUATION TO LEFT OF WORD	Comma or semicolon used incorrectly.
003	T	ILLEGAL USE OF RESERVED WORD-TREATED AS NAME	Reserved word not used in proper division.
004	T	NO SPACE BEFORE LEFT PARENTHESIS	Space must precede (.
005	T	ILLEGAL SPACE BEFORE RIGHT PARENTHESIS	No space allowed before).
006	T	NO SPACE AFTER PERIOD - ASSUMED TO BE PERIOD	No space or numeric after period indicator.
007	E	INVALID CHARACTER(S) IN WORD - TREATED AS NAME	Character is not in the character set or special character.
008	E	DATA-NAME OR LITERAL OR PICTURE GREATER THAN 30 CHARACTERS - TRUNCATED ON THE RIGHT	30 character maximum.
009	E	NON-NUMERIC LITERAL GREATER THAN 255 CHARACTERS -	255 characters maximum.
010	E	) CANNOT BE FIRST CHARACTER OF WORD	Check for bad keypunch.
011	T	) NOT FOLLOWED BY SPACE	Blank or a period must follow).
012	T	NO SPACE FOLLOWING PUNCTUATION	Space must follow punctuation.
013	T	ILLEGAL CHARACTER IN COLUMN 7 - IGNORED	Character other than hyphen or blank in column 7; ignored.
014	E	MISSING QUOTE ON CONTINUATION CARD OF NON-NUMERICAL LITERAL	Literal will be truncated at continuation point.
015	T	NEITHER (NOR) MAY BE CONTINUED	Attempt to continue a formula or subscript from previous card.

No.	Type	Diagnostic	Discussion
016	T	MISSING QUOTE AT END OF NON-NUMERIC LITERAL	Literal is truncated if no continuation on new card.
017	T	FIRST WORD ON NEXT CARD SHOULD BEGIN IN COLUMN 8	Bad A margin begin for a division header, procedure section or paragraph name.
018	T	SEQUENCE NUMBER IN COLUMNS 1-7 BREAKS ON NEXT CARD	Card is misplaced or rearranged cards have not been resequenced.
022	T	NON-CDC RESERVED WORD THIS DIVISION - ACCEPTED AS DATA NAME	
101	T	IDENTIFICATION HEADER NOT FIRST CARD OF PROGRAM	COBOL requires first card in source program to be Identification Division header. If it is not a division header next card will be read.
102	E	BAD WORD OR COMPILER OUT OF SYNCHRONIZATION	Compiler encountered a word it does not recognize or an inappropriate word. Also given for words encountered while compiler is trying to find a key word from which it can resume processing.
103	E	PROGRAM-ID NOT FIRST PARAGRAPH IN ID DIVISION	First item in the Identification Division should be Program ID.
104	T	NO PROGRAM NAME - ASSIGNED IDCOBOL	If no Program ID exists, compiler assigns ID-COBOL as the program name
105	T	WORD 'IS' IS MISSING	Given when IS is missing and required by COBOL. Its absence does not affect compilation.
106	E	BAD SPECIAL NAMES CLAUSE-CLAUSE IGNORED	Indicates one of the following errors occur: 1. The object of the COPY clause is a key word. 2. The clause's DECIMAL POINT is COMMA does not contain the word COMMA. 3. The object of the currency sign is not a non-numeric literal as required. 4. The switch indicated is not an integer 1 through 6. 5. The object of the switch-status is not an acceptable name.

No.	Type	Diagnostic	Discussion
107	E	ILLEGAL USE OF RESERVED WORD FOR NAME	Given when object of COPY clause is a key word rather than acceptable name or integer; entire clause is ignored.
108	E	NO FILES SELECTED	SELECT clause does not have acceptable file name; remainder of cluase is ignored. Compiler scans to end of sentence.
109	E	NO OBJECT OF RENAMING CLAUSE	Object of RENAMING in SELECT clause is not acceptable; renaming option is ignored.
110	C	ASSIGN OPTION IS MISSING - FATAL	No ASSIGN option following SELECT clause. SCOPE system requires assignment of a scope-type file name (implementor name) as object of ASSIGN. This implementor name is assigned to a physical unit through SCOPE object load time. The compiler terminates after PASS 1E.
111	T	WORD 'REEL' OR 'UNIT' IS MISSING-ASSUME REEL	REEL or UNIT is missing from MULTIPLE option; REEL is assumed.
112	T	WORD 'SEQUENTIAL' IS MISSING BUT IS ASSUMED	SEQUENTIAL is missing from the PROCESSING option; assumed by compiler. RANDOM is not presently implemented for PROCESSING option.
113	T	BAD ACCESS CLAUSE - ASSUME SEQUENTIAL	Object of ACCESS option must be SEQUENTIAL or RANDOM; compiler assumes SEQUENTIAL.
114	E	BAD FILE-LIMIT CLAUSE-CLAUSE IGNORED	FILE-LIMIT option does not contain object integer as object; clause is ignored.
115	T	BAD RESERVE CLAUSE - ASSUME NO ALTERNATE AREA	Object of RESERVE option must be NO or integer. Compiler assumes NO ALTERNATE AREAS.
116	E	BAD FILE-CONTROL PARAGRAPH	First clause in FILE-CONTROL paragraph must be COPY or SELECT; compiler scans input to find next key word starting in column 8.

No.	Type	Diagnostic	Discussion
117	E	BAD ITEM ON LIBRARY	Item to be copied from the library is not a SELECT CLAUSE. COPY clause is ignored and compiler scans input same as above.
118	E	BAD RERUN CLAUSE - CLAUSE IGNORED	RERUN clause in I-O-CONTROL paragraph is not correctly specified. Clause is ignored.
119	E	BAD SAME CLAUSE - CLAUSE IGNORED	Object of SAME clause must be a file name. If file name is not acceptable, clause is ignored. However if SAME clause is for SORT area, the result of compilation will be correct since SORT uses the same area for all sort files, and this clause is for documentation only.
120	T	WORD 'FILE' IS MISSING BUT IS ASSUMED	FILE is missing from MULTIPLE FILE clause; has no effect on compilation.
121	E	BAD MULTIPLE FILE CLAUSE - CLAUSE IGNORED	MULTIPLE FILE clause in I-O-CONTROL paragraph contains misplaced or inacceptable file name or integer. Entire clause is ignored.
122	T		None as yet.
123			None as yet.
124	E	FILE-NAME ALREADY SELECTED	File name defined by previous SELECT in the FILE-CONTROL paragraph. This will result in two file tables being defined with same name.
125	T	MISSING DATA RECORD OR REPORT CLAUSE IN FILE DESCRIPTION	A file Description (FD) must contain DATA RECORDS or a REPORT clause, not both. Sort Description (SD) must contain DATA RECORDS clause.
126	C	NOT ALL RENAMED FILES WERE SELECTED - FATAL	One of names that is object of RENAMING in SELECT not selected within this FILE-CONTROL paragraph. Compiler terminates at end of Pass 1E.
127	E	FILE-CONTROL PARAGRAPH IS MISSING	

No.	Type	Diagnostic	Discussion
128	E	FILE-CONTROL PARAGRAPH OUT OF ORDER OR DUPLICATED	
129	T	SORT FILES ALWAYS SHARE THE SAME AREA	SORT uses same area for all files; it is redundant to use SAME SORT area clause in I-O-CONTROL paragraph. Compilation continues normally.
130	C	MISSING, MISPLACED OR DUPLICATED DIVISION - FATAL	Major division or division header is missing. Identification, Environment, or Data Division not in proper sequence. Compiler terminates at end of Pass 1E.
200	E	FEATURE NOT IMPLEMENTED	A feature of DOD COBOL specified card not yet implemented in this COBOL compiler.
201	T	WORD 'DIVISION' MISSING ON HEADER	DIVISION is a required word; compilation continues.
202			None as yet.
203	T	WORD 'SECTION' MISSING ON HEADER	SECTION is a required word; compilation continues.
205	E	REPORT DESCRIPTION FOUND IN NON-REPORT SECTION	RD appeared in section of Data Division other than Report Section.
206	E	BAD LEVEL NUMBER - ITEM IGNORED	Level number is not in accepted range for level numbers; entire sentence is ignored.
207	T	MISSING OR MISPLACED PERIOD	Period is missing or misplaced in sentence or header.
208			None as yet.
209	E	MISSING DIVISION HEADER	Data Division header does not precede section header in Data Division.
210			None as yet.

No.	Type	Diagnostic	Discussion
211	E	NO FILE DESCRIPTION PRECEDING RECORD	First acceptable item in File Section appears to be record description rather than file description. First item in File Section must be an FD or SD item.
212			None as yet.
213	E	PROCEDURE-NAME NOT PRECEDED BY HEADER	Procedure-name has appeared in Data Division. A data name in the Data Division must have one alphabetic character.
214	C	NO SELECT FOR FILE - FATAL	FD encountered in file section has no corresponding SELECT in Environment Division. Since the file table is initially set up by compiler when it encounters SELECT processing of file terminates at end of Pass 1E.
215	E	NO NAME FOLLOWING LEVEL NUMBER	Data name which is required to follow a level number has not been found.
216	E	BAD OR MISSING LABEL CLAUSE-ASSUME OMITTED	Either no LABEL clause appeared in FD description or clause was improperly written.
217	T	WORD 'RECORD' IS MISSING FROM LABEL CLAUSE	
218	T	NUMERIC LITERAL 'ALL' IS ILLEGAL - ASSUME ALPHANUMERIC	ALL in a VALUE clause preceding a numeric literal is not legal; ALL will be honored and the numeric literal will be considered as alphanumeric in class.
219	E	BAD VALUE FOR STANDARD LABEL	An acceptable LABEL clause encountered but VALUE clause for the label is missing. Compiler continues processing and produces a file with label omitted.

No.	Type	Diagnostic	Discussion
220	E	BAD VALUE CLAUSE	VALUE clause in the file description (FD) does not have acceptable literal or data-name as object; clause is ignored.
221	T	WORD OF IS MISSING ON VALUE CLAUSE	OF is missing in VALUE clause. Compilation continues.
222	T	WORD 'RECORD' IS MISSING ON DATA RECORDS CLAUSE	OF is missing in VALUE clause. Compilation continues.
223	E	BAD RECORD CONTAINS CLAUSE	RECORD CONTAINS clause in file description (FD) does not contain acceptable integer or DEPENDING ON option is not followed by key word, record mark, or acceptable name. DEPENDING ON option is ignored.
224	T	WORD 'TO' MISSING	
225	T	WORD 'RECORDS' IS MISSING FROM FILE CONTAINS CLAUSE	
226	T	BAD FILE CONTAINS CLAUSE	FILE CONTAINS clause does not contain acceptable integer; does not effect compilation since clause is presently ignored.
227	E	BAD BLOCK CONTAINS CLAUSE	BLOCK CONTAINS clause does not contain acceptable integer as block size; a size of 64 is assumed.
228	E	BAD NAME IN REDEFINES CLAUSE - CLAUSE IGNORED	The word following REDEFINES is not an acceptable data-name; clause is ignored.
229	E	CLAUSE NOT APPROPRIATE ON SORT FILE	A clause in SD description is not appropriate as a SORT file descriptor. Clause is processed normally but will be ignored by the SORT.
230	E	NO NAME GIVEN IN COPY CLAUSE - ITEM IGNORED	Object of COPY clause not an acceptable name; clause is ignored.

No.	Type	Diagnostic	Discussion
231	E	WORD 'FROM' NOT FOLLOWED BY WORD 'LIBRARY' - 'LIBRARY ' ASSUMED	A COPY clause contains FROM but not LIBRARY. The word LIBRARY is assumed and processing continues.
232	E	WORD 'OF' NOT FOLLOWED BY AN ACCEPTABLE NAME	OF follows a name implying qualification, but qualifying name does not appear. In most cases, the item is discarded.
233			None as yet.
234	E	INDICATED NAME CANNOT BE FOUND ON LIBRARY	Name specified by COPY FROM LIBRARY not found on the library. Ignored.
235	E	INDICATED SIZE TOO LARGE - SET TO MAXIMUM OF 255	Size stated is too large for item. Will be set to 255; maximum size for an elementary item.
236	E	SIZE CLAUSE DOES NOT CONTAIN INTEGER	
237	E	WORD 'LEFT' OR 'RIGHT' MISSING FROM POINT CLAUSE - 'LEFT' ASSUMED	
238	E	INDICATED POINT LOCATION TOO LARGE - SET TO ZERO	Integer in POINT LOCATION clause exceeds 31. Maximum point location is 31 places (left to right).
239	E	POINT CLAUSE DOES NOT CONTAIN INTEGER - SET TO ZERO	
240	T	WORD 'RIGHT' OR 'LEFT' MISPLACED	RIGHT or LEFT should not follow POINT LOCATION clause. Compiler processes the clause as though the word precedes the integer.
241	E	'LEFT' OR 'RIGHT' MISSING FROM SYNCHRONIZED CLAUSE 'LEFT' ASSUMED	
242	E	BAD USAGE CLAUSE - ASSUME DISPLAY	USAGE clause does not contain acceptable word as object. DISPLAY is assumed.

No.	Type	Diagnostic	Discussion
243	E	BAD CLASS CLAUSE - ASSUME ALPHANUMERIC	CLASS clause does not contain acceptable word as object. ALPHANUMERIC is assumed.
244	T	ITEM CAN NOT BE DESIGNATED AS JUSTIFIED LEFT	COBOL does not allow JUSTIFIED LEFT clause. Numeric items are always justified right. Alphanumeric and alphabetic items are normally justified left. The clause is ignored.
245	E	BAD JUSTIFIED CLAUSE	JUSTIFIED was not followed by RIGHT. Clause is ignored.
246	U	BWZ ACCEPTED AS BLANK WHEN ZERO	BWZ has been implemented as a substitute for BLANK WHEN ZERO.
247	T	WORD 'ZERO' ASSUMED FOR BLANK WHEN ZERO	
248	E	BAD ZERO SUPPRESS CLAUSE OR MISPLACED WORD 'ZERO'	ZERO is not followed by SUPPRESS. ZERO SUPPRESS will not result. The clause is ignored.
249	T	WORD 'SIGN' IS MISSING FROM FLOAT CLAUSE BUT IS ASSUMED	SIGN is not encountered in FLOAT clause; it is assumed and compilation continues.
250	E	WORD 'LEAVING' NOT FOLLOWED BY INTEGER - OPTION IGNORED	
251	E	LEAVING INTEGER TOO LARGE - PRECEDING EDITING CLAUSE IGNORED	Integer in LEAVING clause is larger than number of places to left of decimal point, resulting in no suppression, regardless of value moved. (Character suppression is ignored).
252	E	BAD DEPENDING ON NAME	DEPENDING ON option in OCCURS clause does not contain acceptable data-name. Option is ignored.

No.	Type	Diagnostic	Discussion
253	E	WORD 'OCCURS' NOT FOLLOWED BY INTEGER - CLAUSE IGNORED	OCCURS not followed by acceptable integer; clause is ignored.
254	E	WORD 'TO' NOT FOLLOWED BY INTEGER	TO appears after first integer in OCCURS, but is not followed by acceptable integer. First integer is used as OCCURS total.
255	E	VALUE CLAUSE DOES NOT HAVE AN ACCEPTABLE LITERAL	No acceptable literal as object of VALUE clause, ignored.
256	E	WORD 'THRU' NOT FOLLOWED BY A LITERAL	THRU appears in VALUE clause implying a literal but does not appear. The rest of the clause is ignored, and some diagnostics may be repeated until the compiler recognizes a new clause.
257	E	RENAMES ON A NON-66 LEVEL ITEM-IGNORED	Key word RENAMES appeared with item in Data Division not assigned level 66. RENAMES can only appear on level 66 item; item is ignored.
258	E	NO NAME FOLLOWING WORD 'THRU' OR 'RENAMES'	THRU in RENAMES clause, RENAMES not followed by acceptable data-name. First name is used as entire RENAMES range. In second case, item is ignored.
259	E	MISPLACED COPY CLAUSE - ITEM IGNORED	COPY in a Data Division item does not follow REDEFINES clause. Only REDEFINES may precede COPY. No clause may follow the COPY clause.
260			None as yet.
261	T	BAD RANGE CLAUSE	RANGE clause does not have appropriate literal as object of THRU.

No.	Type	Diagnostic	Discussion
262	E	LEVEL NUMBER NOT APPROPRIATE IN THIS SECTION	
263	E	SECTION DUPLICATED OR OUT OF ORDER	Section in Data Division appeared twice, or is out of order. If a section appears twice, results are erroneous. If entire section is out of order, compiler produces proper results.
264			None as yet
:			:
288			None as yet.
289	E	AN ACCEPTABLE NAME DOES NOT FOLLOW LEVEL NUMBER - ASSUME 'FILLER'	An item in a section other than Report does not have acceptable name or FILLER following level number. FILLER is assumed and processing continues.
290	T	RECORD NAME NOT LISTED IN DATA RECORDS CLAUSE	Name of logical record (01) not listed in DATA RECORDS clause or associated file description (FD). Processing continues normally.
291			None as yet.
292	E	BAD SUBSCRIPT - ITEM IGNORED	Subscript in item is not acceptable. Item is ignored.
293	E	LEVEL 77 NOT FIRST IN SECTION - ITEM IGNORED	Level 77 item must appear as the first item in a section. It may not appear in logical record.
294	E	LEVEL 77 IN FILE SECTION - ITEM IGNORED	Level 77 items are illegal in File Section.
295	E	SECTION HEADER NOT FOLLOWED BY ACCEPTABLE LEVEL NUMBER	Section header of FD not followed by acceptable level number. Item is destroyed and further items are destroyed until level number such as FD, 01, or 77 is encountered.

No.	Type	Diagnostic	Discussion
297	E	CONFLICTING SIGNED AND PICTURE CLAUSES - PICTURE ACCEPTED	SIGNED clause for which PICTURE does not have an S. PICTURE clause is accepted; SIGNED clause is ignored.
298	E	CONFLICTING EDITING AND SIGNED CLAUSES - EDITING ACCEPTED	SIGNED clause appeared along with EDITING clause. SIGNED clause is ignored.
299	E	CONFLICTING EDITING AND USAGE CLAUSES - ASSUME USAGE DISPLAY	EDITED item can have DISPLAY usage only.
300	T	EDITING ILLEGAL ON FILLER ITEM - CLAUSE IGNORED	EDITING is illegal on FILLER item; clause is ignored.
301	T	JUSTIFIED CLAUSE ILLEGAL ON NUMERIC ITEM - CLAUSE IGNORED	JUSTIFIED clause is illegal on numeric items; justified right automatically by the compiler. Left justification on numeric items not allowed. Clause is ignored.
302	E	USAGE COMPUTATIONAL REQUIRES NUMERIC CLASS - ASSUME NUMERIC	Class defined is not NUMERIC. The compiler assumes NUMERIC class.
303	T	USAGE COMPUTATIONAL-N IS ALWAYS SYNCHRONIZED	Redundant; computational-n item is always synchronized.
304	E	VALUE ON A REDEFINES OR OCCURS CLAUSE - VALUE IGNORED	
305	E	REDEFINES CLAUSE ILLEGAL ON FILLER ITEM - REDEFINES IGNORED	
306	E	OCCURS CLAUSE ILLEGAL ON FILLER ITEM - OCCURS IGNORED	
307	E	FILLER ILLEGAL ON 77 and 88 - ITEM IGNORED	

No.	Type	Diagnostic	Discussion
308	E	SIZE CLAUSE ILLEGAL ON 88 ITEM - SIZE IGNORED	Size of 88 item is taken from condition variable (or its dominant item).
309	E	OCCURS CLAUSE ILLEGAL ON 88 ITEM - OCCURS IGNORED	
310	E	USAGE CLAUSE ILLEGAL ON 88 ITEM - USAGE IGNORED	
311	E	REDEFINES CLAUSE ILLEGAL ON 88 ITEM - REDEFINES IGNORED	
312	E	EDITING CLAUSE ILLEGAL ON 88 ITEM - EDITING IGNORED	
313	E	VALUE CLAUSE ON 88 ITEM IS MISSING - ITEM IGNORED	
314	E		None as yet.
315	E	OCCURS CLAUSE ILLEGAL ON 77 ITEM - OCCURS IGNORED	
316	E	VALUE CLAUSE ON 77 ITEM IN CONSTANT SECTION IS MISSING - ITEM IGNORED	
317	E	LEVEL 66 IS NOT A RENAMES ITEM - ITEM IGNORED	
318	E	SIZE CLAUSE ON ELEMENTARY LEVEL IS MISSING - ASSUME SIZE = 1	Elementary items must have SIZE or PICTURE clause.
319			None as yet.
320	E	SIGNED CLAUSE ILLEGAL ON GROUP ITEM - SIGNED CLAUSE IGNORED	

No.	Type	Diagnostic	Discussion
321	E	POINT LOCATION CLAUSE ILLEGAL ON GROUP ITEM - POINT LOCATION IGNORED	
322	E	EDITING AND PICTURE CLAUSES ILLEGAL ON GROUP ITEM - CLAUSE IGNORED	
323	E	JUSTIFIED CLAUSE ILLEGAL ON GROUP ITEM - CLAUSE IGNORED	
324	E	SYNCHRONIZED CLAUSE ILLEGAL ON GROUP ITEM - CLAUSE IGNORED	
325	E	LEVEL 66 NOT LAST ITEM IN RECORD- ITEM IGNORED	Level 66 items cannot be imbedded within a logical record. RENAMES item must immediately follow logical record to which it is associated.
326	E	FIRST ITEM IN FILE NOT AN 01 - ASSUME 01	
327	T	QUALIFYING NAME NOT PRECEDED BY 'OF' OR 'IN'	Two names appeared without an intervening OF or IN. Second name will be assumed to be a qualifier of the first.
328	E	RENAMES CANNOT BE ASSOCIATED WITH OTHER CLAUSES - ITEM IGNORED	
329	E	VALUE ON NON-88 IN FILE SECTION - ITEM IGNORED	
330	E	RENAMES ON NON-66 - ITEM IGNORED	
331	T	REDEFINES ILLEGAL ON AN 01 IN FILE SECTION	All logical records within a file automatically redefine each other since they all use same record area.

No.	Type	Diagnostic	Discussion
332	E	DEPENDING ON CLAUSE NOT ALLOWED ON SORT FILE - CLAUSE IGNORED	
333	E	BOTH REPORT CLAUSE AND DATA RECORDS CLAUSE ON SAME FD	Only one of the clauses may appear in an FD description. Compiler action not presently defined.
334	T	COPY CLAUSE DOES NOT END WITH PERIOD - SKIP TO NEXT CARD	
335	T	REQUIRED COMMA IS MISSING BUT IS ASSUMED	
336			None as yet.
:			
356			
500	E	IMPROPER DIVISION HEADER.	Word DIVISION missing. Skip to next card.
501	E	IMPROPER DIVISION HEADER.	Period missing. Skip to next card.
502	T	PROCEDURE NAME MISSING.	Neither SECTION name nor paragraph name for first paragraph in Procedure Division. A standard name is substituted.
503	T	SECTION NAME IS MISSING.	First section in Procedure Division does not have name; a standard name is substituted.
504	E	PREMATURE END OF PROGRAM.	Period missing on last sentence. A period is assumed.
505	E	MISPLACED END.	END found while looking for a new sentence. Compiler skips to section name.
506	E	THIS ERROR SHOULD NOT OCCUR IN FINISHED COMPILER	Subroutine return where none is expected.

No.	Type	Diagnostic	Discussion
507	E	IMPROPER SECTION HEADER.	Period missing on declarative header. Skip to next card.
508	E	SECTION NAME MISSING.	Procedure name missing for the first section in declaratives. Search for section name.
509	E	SECTION NAME IS MISSING.	SECTION missing on first section in declaratives.
510	E	PREMATURE END OF PROGRAM.	No END DECLARATIVES.
511	E	IMPROPER END DECLARATIVES.	DECLARATIVES not found. Skip to next card.
512	E	IMPROPER END DECLARATIVES.	Period missing. Skip to next card.
513	E	PERIOD MISSING.	Period missing after SECTION look for USE.
515	E	PERIOD MISSING.	Period missing from SECTION header. Search for period.
516	T	NULL SECTION.	Text of section is missing.
517	T	NULL SECTION.	Text of section is missing.
518	T	NULL PARAGRAPH.	Text of paragraph missing.
519	E	PERIOD MISSING.	At end of USE sentence.
520	E	PERIOD MISSING.	After EXIT.
521	E	PROCEDURE NAME MISSING.	INCLUDE skipped.
522	E	REPLACEMENT PAIR MISSING.	INCLUDE skipped.
523	E	PERIOD MISSING.	INCLUDE skipped.
524	E	PERIOD MISSING.	Missing at end of imperative or conditional statements.

No.	Type	Diagnostic	Discussion
525	E	PREMATURE END OF PROGRAM.	While processing ON SIZE ERROR.
526	E	PROCEDURE NAME MISSING.	Following OF (could be reserved word).
527	E	PROCEDURE NAME MISSING.	Following THRU.
528	E	CONDITIONAL STATEMENT NOT ALLOWED WITHIN AT END, ON SIZE OR INVALID KEY	
529	E	PERIOD OR ELSE MISSING.	Following last imperative after ON SIZE ERROR, AT END, or INVALID KEY. Skip to period or ELSE.
530	E	VARYING CLAUSE BAD.	Looking for FROM.
531	E	VARYING CLAUSE BAD.	Looking for word after FROM.
532	E	VARYING CLAUSE BAD.	Looking for BY.
533	E	VARYING CLAUSE BAD.	Looking for word after BY.
534	E	VARYING CLAUSE BAD.	Looking for UNTIL.
535	E	VARYING CLAUSE BAD.	Looking for the condition.
536	E	PROCEED CLAUSE BAD.	Looking for TO.
537	E	PROCEED CLAUSE BAD.	Looking for PROCEED.
538	E	PROCEED CLAUSE BAD.	Looking for TO after PROCEED.
539	E	PROCEED CLAUSE BAD.	Looking for second procedure name.
540	E	WITH NO REWIND BAD.	Looking for NO.
541	E	WITH NO REWIND BAD.	Looking for REWIND.

No.	Type	Diagnostic	Discussion
542	E	INPUT CLAUSE BAD.	
543	E	OUTPUT CLAUSE BAD.	
544	E	FILE NAME MISSING.	
545	E	WITH NO REWIND BAD.	Looking for NO.
546	E	WITH NO REWIND BAD.	Looking for REWIND.
547	E	IDENTIFIER MISSING.	After ASCENDING.
548	E	ASCENDING OR DESCENDING MISSING.	In SORT KEY clause.
549	E	IDENTIFIER MISSING.	After COMPUTE.
550	E	FROM OR EQUALS MISSING.	From compute statement.
551	E	IDENTIFIER MISSING.	From ADD corresponding.
552	E	TO MISSING.	From ADD corresponding.
553	E	RESULT MISSING.	From ADD corresponding.
554	E	IDENTIFIER MISSING.	From ACCEPT.
555	E	MNEMONIC NAME UNRECOGNIZABLE.	From ACCEPT.
556	E	PROCEED TO CLAUSE MISSING.	
557	E	FILE NAME MISSING.	At least one file must be closed.
558	E	MNEMONIC NAME UNRECOGNIZABLE.	From DISPLAY.
559	E	CONDITION UNRECOGNIZABLE.	In IF.
560	E	TRUE STATEMENTS MISSING.	In IF.

No.	Type	Diagnostic	Discussion
561	E	"SENTENCE" MISSING FROM "NEXT SENTENCE."	
562	E	STATEMENTS APPEAR AFTER "NEXT SENTENCE."	Before ELSE or period.
563	E	FALSE STATEMENTS MISSING.	In IF SENTENCE missing from NEXT SENTENCE.
564	E	"SENTENCE" MISSING FROM "NEXT SENTENCE."	
565	E	PARAGRAPH NAME MISSING AFTER PERFORM.	
566	E	PARAGRAPH NAME MISSING AFTER THRU.	
567	E	"TIMES" MISSING.	
568	E	CONDITION MISSING AFTER "UNTIL."	
569	E	BAD "VARYING" CLAUSE.	
570	E	BAD "VARYING" CLAUSE.	
571	E	BAD "VARYING" CLAUSE.	
572	E	CLAUSE MISSING AFTER WITH.	
573	E	CLAUSE MISSING AFTER WITH.	
574	E	NO REWIND MISSING.	
575	E	INFORMATION AFTER OPERATIONAL SIGN UNINTELLIGIBLE.	

No.	Type	Diagnostic	Discussion
576	E	BADLY POSITIONED SIGN.	
577	E	BAD QUALIFIER.	
578	E	ALPHA LITERAL MISSING AFTER "ALL."	
579	E	OBJECT OF CONDITION MISSING.	
580	E	SUBSCRIPT TERM MISSING.	
581	E	SECOND SUBSCRIPT TERM MISSING.	
582	E	THIRD SUBSCRIPT TERM MISSING.	
583	E	RIGHT) MISSING.	
584	E	BAD STATEMENTS IN BRANCH OF IF VERBS.	
585	E	BAD TERM AFTER * OR /.	
586	E	BAD EXPONENT.	
587	E	EXTRA RIGHT).	
588	E	MISSING RIGHT).	
589	E	BAD ABBREVIATION TYPE 3.	
590	E	BAD ABBREVIATION TYPE 3.	
591	E	CONDITION MISSING.	
592	E	RIGHT) MISSING.	
593	E	REPORT GROUP MISSING FROM GENERATE STATEMENT.	

No.	Type	Diagnostic	Discussion
594	E	ONE STATEMENT PARAGRAPH CONTAINS MORE THAN ONE SENTENCE.	
595	E	AND NOT FOLLOWED BY OBJECT.	
596		PROCEDURE NAME EXPECTED HERE OR KEYWORD IS MISPELLED.	When compiler cannot recognize a keyword, it looks next for a procedure name, even if previous sentence did not have a period or the word is in column 8.
597		AT END OR INVALID KEY CLAUSE IN READ IS MISSING.	
598		AT END OR INVALID KEY CLAUSE IN RETURN IS MISSING.	
599		A COMPLETE SUBJECT, RELATION, OBJECT IS NOT AVAILABLE TO SUBSTITUTE IN THIS APPARENT ABBREVIATION.	A complete relation must appear earlier in the same sentence.

No.	Type	Diagnostic	Discussion
600		IDENTIFIER MISSING	After AND in SORT KEY.
601		IDENTIFIER MISSING	Second operand in add statement is not an identifier.
602		INCORRECT SYNTAX IN MULTIPLY	
603		IDENTIFIER MISSING	Word immediately after ADD is unrecognizable.
604		IDENTIFIER MISSING	Word after TO is not recognized as identifier or number.
605		GIVING MISSING	Only GIVING allowed after TO numeric-literal.
606		NON DOD COBOL	TO and GIVING in same statement is not DOD COBOL.
607		ARITHMETIC EXPRESSION IS BAD	Literal, identifier, or formula following FROM = or EQUALS is incorrect.
608		ARITHMETIC EXPRESSION IS BAD	
609		IDENTIFIER MISSING	In DISPLAY statement.
610		IDENTIFIER MISSING	In DIVIDE statement.
611		INTO OR BY MISSING	
612		GIVING MISSING	DIVIDE INTO or DIVIDE BY numbers require GIVING.
613		RESULT IDENTIFIER MISSING	After DIVIDE GIVING.
614		IDENTIFIER MISSING	Identifier or number missing after DIVIDE into.
615		IDENTIFIER MISSING	

No.	Type	Diagnostic	Discussion
616		PROCEDURE NAME MISSING	In ENTRY statement.
617		SUBROUTINE NAME MISSING	In ENTER statement.
618		IDENTIFIER MISSING	Following EXAMINE.
619		INCORRECT SYNTAX IN EXAMINE	
620		IMPROPER LITERAL	In EXAMINE.
621		INCORRECT SYNTAX IN GO TO	
622		IDENTIFIER MISSING	In GO TO DEPENDING ON.
623		INCORRECT "REPLACING BY" CLAUSE	
624		IDENTIFIER MISSING	In INITIATE.
625		INCORRECT SYNTAX IN MOVE	
626		INPUT OR OUTPUT MISSING IN OPEN	
627		IN AT END, INVALID KEY OR ON SIZE CLAUSE ARE MISSING OR CONDITIONAL	
628		FILE NAME MISSING	In READ or RETURN or SEEK or SORT.
629		IDENTIFIER MISSING	In READ INTO or RETURN INTO or WRITE FROM or RELEASE FROM.
630		IDENTIFIER MISSING	In SORT KEY clause.
631		SORT KEY MISSING	
632		INCORRECT SYNTAX IN SORT	
633		INCORRECT SYNTAX IN STOP STATEMENT	

No.	Type	Diagnostic	Discussion
634		INCORRECT SYNTAX IN SUBSTRACT	
635		SUBSTRACT CORRESPONDING FROM DATA NAME SERIES IS NOT DOD	
636		INCORRECT SYNTAX IN USE	
637		RECORD NAME MISSING	In WRITE or RELEASE.
640		INCORRECT SYNTAX IN WRITE	
641		INCORRECT SYNTAX IN TERMINATE	
642		WORDS SKIPPED BECAUSE OF INCORRECT SYNTAX	
643		CONDITION NAME USED IMPROPERLY IN A FORMULA	
800		ILLEGAL RESULT FIELD.	
801		ILLEGAL OPERAND.	
802		VALUE TRUNCATED ON RIGHT SO IT WILL FIT INTO FIELD.	
803		VALUE TRUNCATED ON LEFT SO IT WILL FIT INTO FIELD.	
804		COMBINED SIZE OF OPERANDS EXCEEDS 18 DIGITS.	
805		NON-NUMERIC LITERAL WILL BE MOVED TO NUMERIC FIELD, THOUGH INCORRECT.	
806		LITERAL SHIFTED COMPLETELY OUT OF FIELD.	

No.	Type	Diagnostic	Discussion
807		NUMERIC LITERAL WILL BE MOVED TO ILLEGAL FIELD.	
808		NO SUBSCRIPT FOR A SUBSCRIPTED FIELD.	
809		NO SUBSCRIPT WHEN REQUIRED BY OCCURS CLAUSE.	
810		MORE SUBSCRIPTS THAN THERE WERE OCCURS CLAUSES.	
811		ILLEGAL MOVE - TO ALPHABETIC EDITED.	
812		ILLEGAL MOVE - TO NUMERIC.	
813		RECEIVING FIELD CANNOT BE A LITERAL.	
814		SENDING AND RECEIVING FIELDS OF MOVE MAY INTERSECT.	
815		NUMERIC MOVE MAY RESULT IN LEFT TRUNCATION.	
816		NUMERIC MOVE MAY RESULT IN RIGHT TRUNCATION.	

CONVERSION HINTS H

These conversion hints attempt to point out the areas where conversion problems are most often encountered or where a significant amount of the effort in any conversion is spent. The following comments apply to virtually all programs from all machines, but are aimed specifically at conversion from the CONTROL DATA 3600/3800 machines to the CONTROL DATA 6400/6500/6600 machines.

Each program will require the following changes:

1. All control cards and end-of-file cards must reflect 6000 usage.
2. The source computer and object computer statements must state the computer number, 6400, 6500, or 6600.
3. In the SPECIAL-NAMES Section, names having special significance as defined by other compilers must be changed to reflect their new names. Example: SYSTEM-OUTPUT-TAPE used by 3600/3800 becomes OUTPUT for 6400/6500/6600.
4. The SELECT statement must refer to implementor names having one to seven characters conforming to SCOPE restrictions.
5. Generating several files onto a single output tape will require unique names which are assigned by the request control cards.
6. JUSTIFIED LEFT is illegal in 6400/6500/6600.
7. ENTER statements will probably need changing to reflect the machine language subroutine to be used.
8. The data-name option of labels must be used in place of the term NON-STANDARD.

A further problem arises when a COBOL source program written for a word-oriented computer is to be input to a character-oriented computer such as the 6000 series computers. In the 6000 computers, the number of characters in a record need not be a multiple of the number of characters in a machine word. The 6000 series computers can transfer the precise number of characters defined in the Data Division. However, in many machines, including the 3600/3800, a record must be a multiple of a machine word; and the smallest unit that can be read or written is a word. If an entry in the Data Division does not use all characters of the last word on a data record, the whole word will be output by a word-oriented machine. When this record is read back into a word-oriented machine, the extra characters that filled up the last word are ignored. However, if such a record is read into a character-oriented machine, such as the 6000 computers, the extra filler characters which were not part of the COBOL source language would be read as the beginning of the succeeding record.

The solution to this problem is to use the COBOL word FILLER to define the full record to the precise length of the machine word if the source machine is word-oriented.

For example:

If computer A is word-oriented, eight characters per word, a data record should be defined as follows:

```
01  DATA-ITEM-1
    03  WORD-1 PICTURE X(10).
    03  WORD-2 PICTURE X(3).
    03  FILLER PICTURE X(3).
```

DATA-ITEM-1 takes up two full computer words of computer A and can be read by a character-oriented machine with complete accuracy. If FILLER were not included, the next record would contain three characters of unknown content when read by computer B, a character-oriented machine, 10-character-per-word.

Synchronized fields in a record output from one machine and input to another may cause similar conversion problems.

AUTOMATIC PROCESSING OF MULTI-REEL FILES

To process multi-reel files without difficulty, a special feature added to the 6000 COBOL processor differentiates between end-of-reel and end-of-file on input tapes. This feature looks for a special identifier in the end-of-reel label. The identifier is given by the value in the statement: END-OF-REEL-IDENTIFIER IS data-name. Depending on the source machine, the identifier represents the number of characters that allow the COBOL program to differentiate between an end-of-reel and an end-of-file.

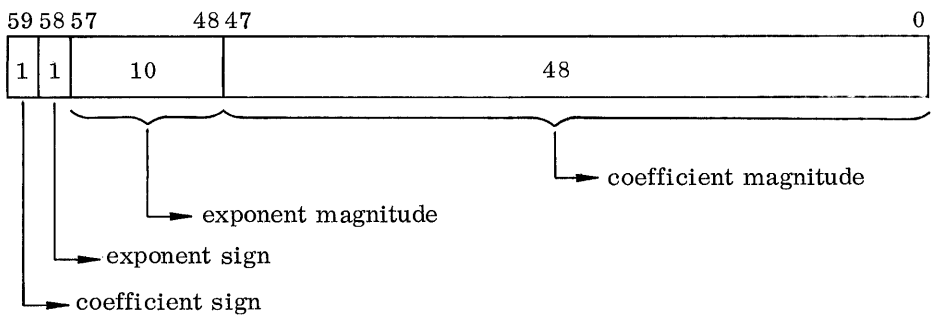
For example, the difference between EOR and EOF. If the programmer sets the data-name to the value EOR, when the program encounters a tape mark at the end of a reel, it will read the following record and compare the first three characters to the value EOR. If the value does compare, the reel is assumed to be an end-of-reel rather than an end-of-file; and an end-of-reel process is initiated with attendant tape swapping. Finally, upon encountering a tape mark, the following comparison will fail; and the assumption will be made that end-of-file has been encountered. In this manner, one-shot on-line conversion can be done without making a special program merely to copy large files. The structure of the files can be read and converted during a regular production run simply by changing a few statements in the original COBOL program.

NUMBER REPRESENTATION I

This appendix contains a description of the 6000 series floating point format and number representation. In addition, it contains the specific format and number representation for the three possible COBOL usages when the class is numeric: DISPLAY or COMPUTATIONAL usage are represented by display code integers, COMPUTATIONAL-1 is represented by unnormalized biased floating point numbers, and COMPUTATIONAL-2 by normalized floating point numbers.

Floating Point Word Format

The 6000 Series 60-bit floating point format is shown below:



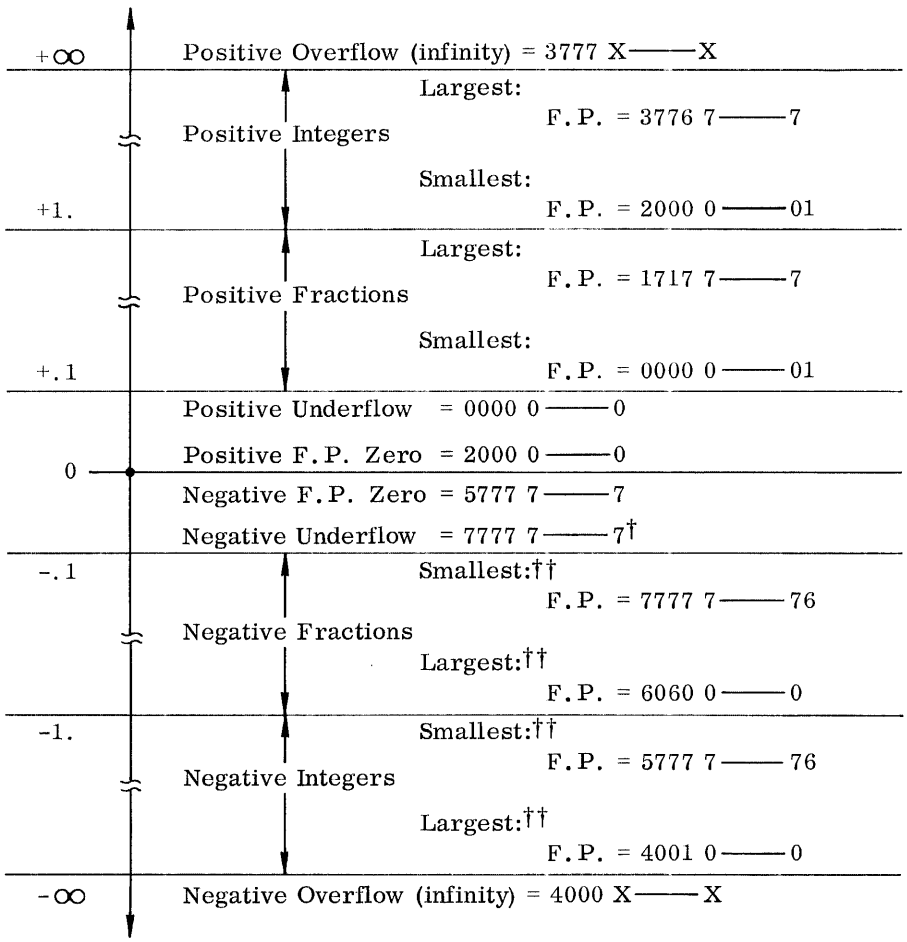
The lower 48 bits are reserved for expressing the magnitude of the coefficient. The computer assumes the binary point to be to the right of the coefficient, thereby providing a 48-bit integer coefficient which is equivalent to about 15 decimal digits. Bit 2^{59} is used to express the sign of the coefficient. The following rules apply to signed coefficients:

Sign	2^{59} =	Magnitude (bits 0-47)
Positive	0	True (uncomplemented) form
Negative	1	Complement form

The table below summarizes the configurations of bits 2^{58} and 2^{59}, and the implications, regarding signs, of the possible combinations.

2^{59}	2^{58}	Coefficient Sign	Exponent Sign
0	1	Positive	Positive
0	0	Positive	Negative
1	0	Negative	Positive
1	1	Negative	Negative

6000 SERIES FLOATING POINT REPRESENTATION



† The machine packs all zeros (positive underflow) for this case.
†† In absolute value.

COBOL USAGE

COMPUTATIONAL-1 items of less than 15 decimal digits in size will be expressed as binary, fixed exponent, integers in an unnormalized floating point format. For example, an item with PICTURE IS 99V99 and VALUE IS 010.000 would appear in memory as 2000 0000 0000 0000 1750. Notice that this is 1000 decimal, it is an integer, it is in unnormalized floating point format, and it occupies one computer word. The exponent of 2000 indicates that the decimal point follows the 1750. The exponent bias is 2000 octal.

Assuming the same item is COMPUTATIONAL-2 the following memory word represents the item: 1731 7650 0000 0000 0000. This is a normalized floating point number. Normalized means that the most significant bit of the integer is moved left to bit 47. Each shift left causes the exponent, which started at 2000, to be reduced by one. The number is completely described using the exponent, coefficient magnitude, and sign of coefficient (bit 59).

If the size of a COMPUTATIONAL-1 item reaches or exceeds 15 decimal digits, an additional computer word is used to hold the number in the double precision, normalized, floating point format. If the number shown above (2000 0000 0000 0000 1750) had exceeded 14 decimal digits it would appear in memory as follows:

word 1 1731 7650 0000 0000 0000

word 2 1651 0000 0000 0000 0000

The table below shows the particular number representations tabulated by the USAGE clause. The largest representation in one usage is not the same as the largest representation in another usage. Numbers carried in display code (COMPUTATIONAL) are assumed positive unless specifically defined as carrying a sign by the SIGNED clause or the editing character S.

Number	COMPUTATIONAL (display code integers only)	COMPUTATIONAL-1 (unnormalized biased floating point)	COMPUTATIONAL-2 (normalized floating point)
	59 bits 0	59 47 bits 0	59 47 bits 0
∞	(not applicable)	3777 (not applicable)	3777 (not applicable)
largest positive	4444 — 4444	2000 7 — 7	3776 7 — 7
largest positive -Δ	4444 — 4443	2000 7 — 76	3776 7 — 76
largest positive - 2Δ	4444 — 4442	2000 7 — 75	3776 7 — 75
⋮	⋮	⋮	⋮
2	33 — 3302	2000 0 — 02	1721 40 — 0
1 + 2Δ	33 — 3336	2000 0 — 03	1720 40 — 02
1 +Δ	33 — 3335	2000 0 — 02	1720 40 — 01
1	33 — 3334	2000 0 — 01	1720 40 — 0
1 - Δ	-	-	1717 7 — 7
1/2	-	-	1717 40 — 0
1/4	-	-	1716 40 — 0
1/8	-	-	1715 40 — 0
+ 0	3333 — 3333	2000 0 — 0	2000 0 — 0
- 0	-	5777 7 — 7	5777 7 — 7
- 1/8	-	-	6062 37 — 7
- 1/4	-	-	6061 37 — 7
- 1/2	-	-	6060 37 — 7
- 1 +Δ	-	-	6060 0 — 0
- 1	33 — 3312	5777 7 — 76	6057 37 — 7
- 1 - Δ	33 — 3313	5777 7 — 75	6057 37 — 76
- 1 - 2Δ	33 — 3314	5777 7 — 74	6057 37 — 75
- 2	33 — 3313	5777 7 — 75	6056 37 — 7
⋮	⋮	⋮	⋮
largest negative + 2Δ	44 — 4420	5777 0 — 02	4001 0 — 02
largest negative +Δ	44 — 4421	5777 0 — 01	4001 0 — 01
largest negative	44 — 4422	5777 0 — 0	4001 0 — 0
-∞	(not applicable)	4000 (not applicable)	4000 (not applicable)

Δ is assumed the smallest positive increment for the particular representation.

Examples:

No. of Digits	Value	Octal Dump Format if Usage Is	
		COMPUTATIONAL-1	COMPUTATIONAL-2
4	10.00	2000 0000 0000 0000 1750	1731 7640 0000 0000 0000
4	1000.	2000 0000 0000 0000 1750	1731 7640 0000 0000 0000
15	00.50	1717 4000 0000 0000 0000 1637 0000 0000 0000 0000†	1717 4000 0000 0000 0000
14	.25000000000000	2000 0043 2254 7242 4000	1716 4000 0000 0000 0000
3	123.	2000 0000 0000 0000 0173	1726 7540 0000 0000 0000
5	32768	2000 0000 0000 0010 0000	1737 4000 0000 0000 0000
Negative numbers are formed by taking the one's complement of positive numbers			
5 digits	-32768	5777 7777 7777 7767 7777	6040 3777 7777 7777 7777

†Depending upon machine operation, the low order part of a COMPUTATIONAL-1 item may have other equivalent representations. It may be normalized. Zero may have another representation.

Entity	Name	Entry Point	Content	Conditions for Presence
Card	Overlay	N/A	OVERLAY (COBCODE, 0,0)	
Relocatable Deck	D.COMON	D.COMON	TALLY FILE-LABEL POINTERS TO LISTS FET LIST REPORT LIST	Not present in subcompilations.
Labeled Common Relocatable Deck	CCOMMON	CCOMMON	COMMON STORAGE FIELDS	Not present in subcompilations.
Relocatable Deck	Implementor-name	Implementor-name	FET Record Storage Area CIO buffer open indicator	Not present in sub-compilations. (One for each file)
Relocatable Deck	Report-name	Report-name	Encoded Report Sum, line, page counters Report group output areas	Not present in sub-compilations. (One for each report)
Blank Common		N/A	CIO buffer areas.	Not present in subcompilations.
Relocatable Deck	C.section-name	As defined by ENTRY's.	Procedure division sections cross-reference tables.	Always present for any compilation.
Card	OVERLAY	N/A	OVERLAY (COBCODE, n,0)	Section with priority n + 50.
Relocatable Deck	C.section-name	As defined by ENTRY's.	Procedure division sections cross-reference tables.	Section with priority n + 50.
EOF mark	N/A	N/A	Tape mark.	Will be written over and deleted by any ensuing compilations.

INDEX

ACCEPT 4-18
ACCESS MODE 2-4
ACTUAL KEY 2-4, 6; 6-2
ADD 4-19
*ADD 6-30
ADD CORRESPONDING 4-20
ALL A-7
ALTER 4-22
AND 4-8; A-9
Arithmetic expressions 4-14
Arithmetic statements 4-15
ASCENDING KEY, sort 4-56
ASSIGN TO 2-4, 5
Assigned value 3-5
AT END
    read 4-52
    return 4-54
AUTHOR 1-1

Binary output from compiler J-1
BLANK WHEN ZERO 3-11, 16; 5-9
BLOCK CONTAINS 3-9, 13

*CANCEL 6-30
*CATALOG 6-28
CF 5-8, 12
CH 5-8, 12
CHECK PROTECT 3-11, 15; 5-9
CLASS 3-11, 14; 5-8, 9, 10
Clause editing 3-11, 15
CLOSE 4-23
CURRENCY SIGN 2-23; 3-29, 31
COBOL
    character set A-1
    coding sheet A-9, 11
    collating sequence B-1
    control card 6-12
    diagnostics G-1
    display code B-1
    language A-1
COBOL reserved words A-6, list C-1
COBOL source library 6-24
CODE 5-4, 5
CODING SHEET A-9, 11
COLUMN 5-9, 16
COMMON-STORAGE SECTION 3-1, 6
Common storage 6-15
COMPILATION 6-10
*COMPILE 6-30
Compiler diagnostics G-1
COMPUTATIONAL-1 3-46
    intermediate results D-1
    number representation I-3
COMPUTATIONAL-2 3-46
    intermediate results D-1
    number representation I-3
COMPUTE 4-24
    intermediate results D-1
CONFIGURATION SECTION 2-1, 2
Condition name 3-4; A-4
Conditional statements 4-5
Conditions 4-5
    class condition 4-7
    compound conditions 4-8
    condition-name condition 4-7
    implied elements 4-9
    nested conditional statements 4-11
    relational condition 4-5
    sign condition 4-7
    simple relational condition 4-5
    switch-status-name condition 4-8
Connectives A-8
CONTROL(S) 5-4, 5
CONTROL FOOTING 5-8, 12
Control groups 5-2
CONTROL HEADING 5-8, 12
Control hierarchy 5-2
CONSOLE 2-2, 3
CONSTANT SECTION 3-1, 7
Constants
    figurative 3-5; A-6
    named A-6

Conversion hints H-1
COPY 2-2, 4; 3-9, 17; 5-4, 10
*COPY 6-28
COPYCYL 6-26
CURRENCY SIGN 2-2, 3

DATA DIVISION 3-1
    clauses 3-12
Data names 3-4; A-3
DATA RECORDS 3-9, 10, 19
DATE-COMPILED 1-1
DATE-WRITTEN 1-1
DE 5-8, 13
*DECK 6-28
DECIMAL-POINT IS COMMA 2-2, 3
DECLARATIVES 4-1, 2
*DELETE 6-29
DESCENDING KEY (SORT) 4-56
DETAIL 5-8, 13
Diagnostics, COBOL compiler G-1
Display code B-1
DISPLAY, USAGE IS 3-46; 5-8, 9
DISPLAY verb 4-25
DIVIDE 4-26

*EDIT 6-29
Edit control cards 6-29
Editing clauses 3-28; 5-10
EDITSYM 6-27
Elementary item 3-2
ELSE (IF) 4-32
*END 6-28
ENTER
    verb 4-27
    calling sequence E-1
ENTRY 4-28; 6-15
Entry 3-2
Entry points 6-15
ENVIRONMENT DIVISION 2-1
EQ 4-6
EQUAL(S) 4-6
Equipment assignment 6-22
EXCEEDS 4-6
EXAMINE 4-29
EXECUTE 6-21
Execution, control cards 6-19, 21
EXIT 4-30

FET 6-3
Figurative constant 3-5; A-6
FILE CONTAINS 3-9, 10, 19
FILE-CONTROL 2-1, 4
File description entry 3-8
    FD format 3-9
    SD format 3-10
    specification 3-10
File environment table 6-3
File format 6-1
File header label 6-6, 8
File index 6-2
FILE-LIMIT(S) 2-4, 6
File name
    SCOPE 6-1
    COBOL A-4
FILE SECTION 3-1, 5; 5-3
File trailer label 6-6, 8
FILLER 3-4
FIRST DETAIL 5-4, 6
FLOAT CURRENCY SIGN 3-11, 15; 5-9
FLOAT DOLLAR SIGN 3-11, 15; 5-9
Floating point
    format I-1
    representation I-2
FOOTING 5-4, 6

GENERATE 5-22
GO TO 4-31
GQ 4-6
GR 4-6
GREATER 4-6
GREATER-EQUAL 4-6
GROUP INDICATE 5-9, 17
Group item 3-2

HEADING 5-4, 6
HIGH-VALUE(S) A-7

IDENTIFICATION DIVISION 1-1
Identifiers A-5
IF 4-32
Imperative statements 4-4
Implementor-name 2-23; A-4
Implied elements 4-9
INCLUDE 4-33

Independent item 3-3
Initial value 3-5
INITIATE 5-21
INPUT PROCEDURE (SORT) 4-56
Input/output control 6-1
INPUT-OUTPUT SECTION 2-1, 4
*INSERT 6-29
Insertion characters 3-28
INSTALLATION 1-1
Intermediate results D-1
INVALID KEY
    (READ) 4-52
    (WRITE) 4-62
I-O-Control 2-1, 7
Item
    group 3-2
    elementary 3-2
    independent 3-3

Job card 6-19
JUSTIFIED 3-11, 20; 5-9

Label record formats 6-7
LABEL RECORDS 3-9, 21
LAST DETAIL 5-4, 6
LEAVING 3-11, 15
LESS 4-6
LESS-EQUAL 4-6
Level number 3-3
    special level numbers 3-3
LINE NUMBER 5-8, 9, 14
LINE-COUNTER 5-2; A-8
Literal 3-5; A-5
LOAD 6-21
LOW-VALUE(S) A-7
LOWER-BOUND(S) A-7
LQ 4-6
LS 4-6

Mass storage I/O, sample program F-10
MOVE 4-35
    matrix of valid MOVES 4-41
MOVE CORRESPONDING 4-40
MULTIPLE FILE 2-7, 8
MULTIPLE REEL 2-4, 6
MULTIPLY 4-42

Name index 6-2
Named constants A-6
NEXT GROUP 5-8, 15
NEXT PAGE 5-14, 15
NEXT SENTENCE (IF) 4-32
NGR 4-6
NLS 4-6
Non-numeric literal IS 2-2, 4
Non-standard labels 3-22; 6-9
NOT 4-6, 8
Notation used in manual ix
NOTE 4-43
NQ 4-6
Number index 6-3
Number representation I-1

OBJECT-COMPUTER 2-1, 2
OCCURS 3-11, 23; 3-35; 3-44
OFF STATUS 2-2, 3
OH 5-8, 12
Omitted label 3-22; 6-10
ON SIZE ERROR option 4-16
ON STATUS 2-2, 3
OPEN 4-44
OPTIONAL 2-4, 5
OR 4-8
OTHERWISE (IF) 4-32
Output from compilation 6-10, 11
OUTPUT PROCEDURE (SORT) 4-56
OV 5-8, 12
OVERFLOW FOOTING 5-8, 12
OVERFLOW HEADING 5-8, 12
OVERLAY 6-15
Overlay loading 6-15
Overlay segments 4-2; 6-15

PAGE 5-4
PAGE FOOTING 5-8, 12
PAGE HEADING 5-8, 12
PAGE LIMIT 5-4, 6
PAGE-COUNTER 5-2; A-8
Page/overflow condition 5-2
PERFORM 4-46
PF 5-8, 12
PH 5-8, 12
PICTURE 3-10, 24; 5-9, 10
    editing 3-28; 5-10

POINT 3-11, 34; 5-9, 10
POSITION 2-7, 8
Priority numbers 4-3
PROCEDURE DIVISION 4-1
    statements 4-17
Procedure names A-3
PROCESSING MODE 2-4, 7
Processor-directing statements 4-5
Program call card 6-21
Program execution control cards 6-21
PROGRAM-ID 1-1
Program identification 6-14
Punctuation A-9

Qualifiers A-4
QUOTE(S) A-7

RANDOM MODE 2-4, 6
RD 5-6, 7
READ 4-52
RECORD CONTAINS 3-9, 10, 35
Record description entry 3-10
    formats 3-11
    specification 3-11
RECORD-MARK A-7
RECORDING MODE 3-9, 36
REDEFINES 3-10, 11, 36
RELEASE 4-54
REMARKS 1-1
RENAMES 3-11, 37
RENAMING 2-4, 5
REPLACING
    (EXAMINE) 4-29
    (INCLUDE) 4-33
Replacement characters 3-30
REPORT(S) 3-9; 5-4
Report description entry 5-4
REPORT FOOTING 5-8
Report group 5-1
Report group description entry 5-8
REPORT HEADING 5-8, 11
REPORT SECTION 3-1, 8; 5-4
Report writer
    description 5-1
    sample program 5-24
REQUEST 6-22
RERUN 2-7, 8
RESERVE ALTERNATE AREAS 2-4, 6
Reserved word list C-1
RESET 5-9, 20
*RESTORE 6-29
RETURN 4-54
RF 5-8, 11
RH 5-8, 11
ROUNDED option 4-16
Rules for arithmetic verbs 4-15

SAME 2-7, 8
Sample programs F-1
Sample program
    COBOL F-1
    mass storage I/O F-10
    report writer 5-24
    SORT F-6
    subcompile F-13
SCOPE 6-1
SCOPE files 6-1
SECTION 4-1, 2
Sections 3-1, 5
SECURITY 1-1
SEEK 4-55
Segmentation 4-2
SELECT 2-4, 5
SELECTED 5-12, 26
Separators A-12
SEQUENCED 3-9
SEQUENTIAL MODE 2-4, 6
SIGNED 3-11, 37; 5-9, 10
SIZE 3-10, 39; 5-8, 9, 10
SIZE-CLASS-USAGE 3-40
SORT
    verb 4-56
    sample program F-6
SOURCE 5-9
SOURCE IS SELECTED 5-18
Source program listing 6-11
SOURCE-COMPUTER 2-1, 2
SOURCE-SUM-VALUE 5-18
SPACE(S) A-7
Special registers A-11
Special system files 6-1
SPECIAL-NAMES 2-1, 2
Standard labels 3-21; 6-6
Statements and sentences 4-4
STOP 4-57

SUB 6-13
Subcompile capability
    description 6-14
    execution 6-16
    sample program F-13
Subscripts 3-44
SUBTRACT 4-58
SUBTRACT CORRESPONDING 4-59
SUM 5-9, 18, 19
SWITCH 2-2, 3
SYMBOLIC KEY 2-4, 6; 6-2
SYNCHRONIZED 3-11, 41

Tables 3-43
TALLY 4-29; A-8
TALLYING 4-29
TERMINATE 5-23
Text decks 6-24
THEN 4-32; A-9
TYPE 5-8, 11

UPPER-BOUND(S) A-7
USAGE 3-11, 46; 5-8, 9
USE 4-60; 6-9
USE BEFORE REPORTING 4-60; 5-23

VALUE 3-11, 48; 5-9, 10
VALUE OF ENDING-TAPE-LABEL 3-9, 21; 6-11
VALUE OF IDENTIFICATION 3-9, 21
Volume header label 6-6, 7
Volume trailer label 6-6, 8

Words A-3
WORKING-STORAGE SECTION 3-1, 6
WRITE 4-62

ZERO(S) (ES) A-7
ZERO SUPPRESS 3-11, 15; 5-9

CONTROL DATA
CORPORATION

COMMENT AND EVALUATION SHEET
6400/6500/6600 COBOL
Reference Manual

Pub. No. 60191200 June, 1967

THIS FORM IS NOT INTENDED TO BE USED AS AN ORDER BLANK. YOUR EVALUATION OF THIS MANUAL WILL BE WELCOMED BY CONTROL DATA CORPORATION. ANY ERRORS, SUGGESTED ADDITIONS OR DELETIONS, OR GENERAL COMMENTS MAY BE MADE BELOW. PLEASE INCLUDE PAGE NUMBER REFERENCE.

FROM NAME: _____

BUSINESS ADDRESS: _____

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA

FOLD FOLD

STAPLE STAPLE

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

Pub. No. 60191200

Litho in U.S.A.