3300 3500

COMPUTER SYSTEMS

USASI COBOL/ MASTER

REFERENCE MANUAL

CONTROL DATA
CORPORATION

60229400

REVISION RECORD

REVISION	NOTES
(2-6-69)	Original printing.

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

Pub. No. 60229400
February 1969

Address comments concerning this manual to:

Control Data Corporation
Software Documentation
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual.

PREFACE

The features of USASI COBOL/MASTER specified in this manual conform to the United States of America Standards Institute (USASI) COBOL Standard. The USASI specifications define a Nucleus for internal processing and eight functional processing modules: Table Handling, Sequential Access, Random Access, Random Processing, Sort, Report Writer, Segmentation, and Library. The Nucleus is divided into two levels, low and high.

USASI COBOL/MASTER is implemented at the high level of the Nucleus. Of the modules, only Random Processing and Segmentation are not implemented. All other modules are implemented at the high level. The Report Writer module can be omitted from the COBOL library. USASI COBOL/MASTER runs under control of the MASTER operating system.

Additional features of COBOL are implemented in USASI COBOL/MASTER to facilitate operating and to support system interfaces. The inclusion of COBOL features over and above the USASI standard is not prohibited by the standard; however, an installation option is available to diagnose all non-USASI features as errors at compile time.

The Business Data Processing Unit (3312) must be present to compile and execute USASI COBOL/MASTER programs on a CONTROL DATA® 3300 computer. The CONTROL DATA® 3500 computer comes equipped with the Business Data Processing hardware.

This manual is organized in the traditional manner of Identification, Environment, Data, and Procedure Divisions with the exception of the Report Writer and Library modules which are described in separate chapters.

ACKNOWLEDGMENT

INTRODUCTION

INTRODUCTION TO COBOL

COBOL (Common Business Oriented Language) is a programming language which uses English terms to simplify the programming of business data processing problems. This manual presents the details and rules for writing a program in USASI COBOL/MASTER for the CONTROL DATA® 3300 and 3500 computers. Familiarity with basic COBOL is desirable, but not essential. This introduction contains a brief outline of the COBOL language elements, program structure and content. Used in conjunction with the rest of the manual, it provides the new user with an introduction to the fundamentals of COBOL programming.

LANGUAGE ELEMENTS

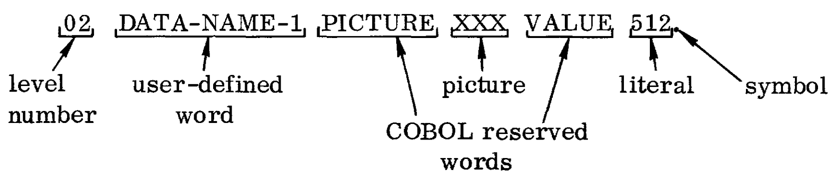
The COBOL language is made up of the following elements:

- COBOL reserved words
- User-defined words
- Literals
- Level numbers
- Symbols
- Pictures

The user composes user-defined words, literals, and pictures according to rules that govern the choice and arrangement of characters. Within the limits imposed by the rules, it is virtually possible to compose an infinite number of words, literals, and pictures.

The COBOL reserved words, symbols, and level numbers are presented in fixed sets from which the user selects what he needs. He cannot invent new reserved words, symbols, or level numbers and must use these elements according to the rules of the language.

The sample COBOL source program entry below shows all six elements:



```
02 DATA-NAME-1 PICTURE XXX VALUE 512.
```

COBOL Reserved Words

Over 250 English words and abbreviations have been set aside as COBOL reserved words. (Appendix C contains a complete list of these reserved words.) Special meanings have been assigned to these words; the user

- Does not define reserved words
- Cannot change the meaning of a reserved word
- Cannot add words to the reserved word list
- Cannot substitute other words for those on the list
- Must not alter or misspell reserved words
- May use reserved words only for specified purposes

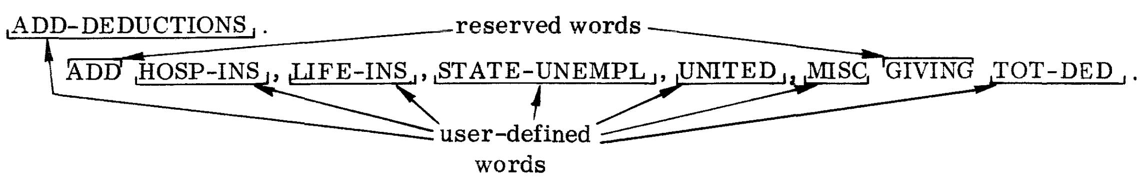
The reserved words have various purposes as defined in the reference formats. They may identify program units, such as SECTION, PROCEDURE, WORKING-STORAGE. They may identify parts of entries, such as VALUE, PICTURE, USAGE. They may specify actions like WRITE, PERFORM, DIVIDE. They may have specific functional meanings such as NEGATIVE, COMPUTATIONAL, EQUAL. As figurative constants, they can represent specific data values: ZERO, SPACES, LOW-VALUE. (A complete list of figurative constants is given in appendix B.)

User-Defined Words

The user must supply words that name data items, data conditions, and procedures. These words have no preassigned meaning, but they must be defined within the program in which they are used. The formation of these words is governed by the following rules:

- A word may be up to 30 characters long.
- It may contain letters (A through Z), digits (0 through 9) and the hyphen (-).
- Only procedure names may be entirely numeric; all other words must contain at least one letter.
- A word may neither begin nor end with a hyphen.
- Spaces (blanks) must not appear in a word.
- It must not be spelled exactly like a reserved word.

The example below contains user-defined words as well as reserved words and symbols.



```
ADD-DEDUCTIONS.
    ADD HOSP-INS, LIFE-INS, STATE-UNEMPL, UNITED, MISC GIVING TOT-DED.
```

Appendix B contains a detailed description of the formation of different types of user-defined words such as data-names, condition-names, procedure-names, and so forth.

Literals

Literals are a special case of user-defined words. They are actual values used in the program and as such are self-defining. Literals may be numeric or non-numeric.

A numeric literal is a string of digits that may include a plus or minus sign, and a decimal point. Its value is the quantity represented by the characters in the literal. Every numeric literal is classed as a numeric item.

A non-numeric literal is a string of one to 120 characters enclosed in quotation marks from the computer's character set (appendix B). It may contain reserved words and spaces but not the quotation mark. The value of a non-numeric literal is the string of characters excluding the quotation marks. Non-numeric literals are classed as alphanumeric items.

Rules governing the formation of literals are given in appendix B.

The following examples show the use of literals in an entry:

```
DIVIDE 1.8 INTO CONVERTED-TEMP1 GIVING TEMP-2.
         numeric literal

77  HEADING-A PICTURE X(10) VALUE IS "COBOL LIST".
                                            non-numeric literal
```

Level Numbers

Level numbers (01 through 49, 66, 77, and 88) are used in entries that assign names to data items and data values. They designate the level of the entries relative to each other.

Numbers 01 through 49 designate the hierarchy of data entries within records. 01 is always assigned to the record itself; 02 through 49 are assigned to items within the record. Level numbers need not be consecutive but must be ordered so that the higher the number, the lower the entry in the hierarchy.

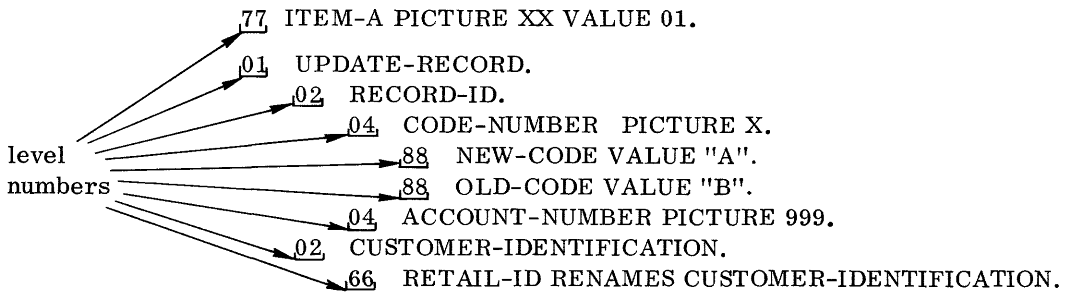
Level number 66 designates an entry that renames a previously defined entry. It is always used with the reserved word RENAMES.

Level number 77 designates an independent data item, one that is not part of a record.

Level number 88 is used with entries that assign condition names to specific values a data item may assume.

Whenever a level number is used, it is the first element in the entry.

The entries below show how level numbers are used:



```
77 ITEM-A PICTURE XX VALUE 01.
01  UPDATE-RECORD.
    02  RECORD-ID.
        04  CODE-NUMBER  PICTURE X.
            88  NEW-CODE VALUE "A".
            88  OLD-CODE VALUE "B".
        04  ACCOUNT-NUMBER PICTURE 999.
    02  CUSTOMER-IDENTIFICATION.
        66  RETAIL-ID RENAMES CUSTOMER-IDENTIFICATION.
```

Symbols

Symbols are special characters which have specific meanings for the compiler. Symbols are used in punctuation, as operators in arithmetic expressions, as relational operators in conditions, and as editing symbols in pictures. The COBOL Character Set in appendix B lists all the symbols available to the COBOL programmer. Punctuation rules are given in appendix B, the rules for arithmetic and conditional operators in chapter 4, and picture symbols in chapter 3.

Pictures

Pictures describe such characteristics of data items as:

- Size of item
- Class of item: numeric, alphabetic, or alphanumeric
- Whether the item is signed
- Position of an assumed decimal point
- Editing (deletion, insertion, or replacement of characters) to be performed on the item

Each picture is a string of from 1 to 30 characters. Pictures are composed of the characters listed under PICTURE in chapter 3. In general, the number of characters defines the size of the item and the character itself defines the class: X=alphanumeric, 9=numeric, A=alphabetic. A picture may be abbreviated by enclosing an integer in parentheses to show the size immediately after the character defining the class. For instance, X(20) means the same as 20 X's in a row; both mean that the item consists of 20 alphanumeric characters.

The word PICTURE or the words PICTURE IS or the abbreviation PIC always precedes the picture itself in an entry.

Examples of pictures in entries:

```
02  FILLER PICTURE X(8).
                             pictures
02  STOCK-ITEM PIC 999.
```

PROGRAM STRUCTURE

A COBOL source program is composed of entries organized into divisions, sections, and paragraphs. In general, a division is made up of sections, and a section is made up of paragraphs.

Divisions

All COBOL programs consist of four divisions each with a fixed name: IDENTIFICATION, ENVIRONMENT, DATA, and PROCEDURE. The divisions always appear in that order in a program.

The beginning of each division is marked by a division header entry consisting of the name of the division followed by the word DIVISION and a period. The division header always appears on a line by itself.

Sections

The inclusion of sections depends on the particular division. There are no sections in the Identification Division. The Environment and Data Divisions always have sections and these sections have fixed names. Sections are optional in the Procedure Division, and when sections are included, the section names are supplied by the user.

Each section is identified by a header entry consisting of the section name followed by the word SECTION and a period. A section header usually appears on a line by itself.

Paragraphs

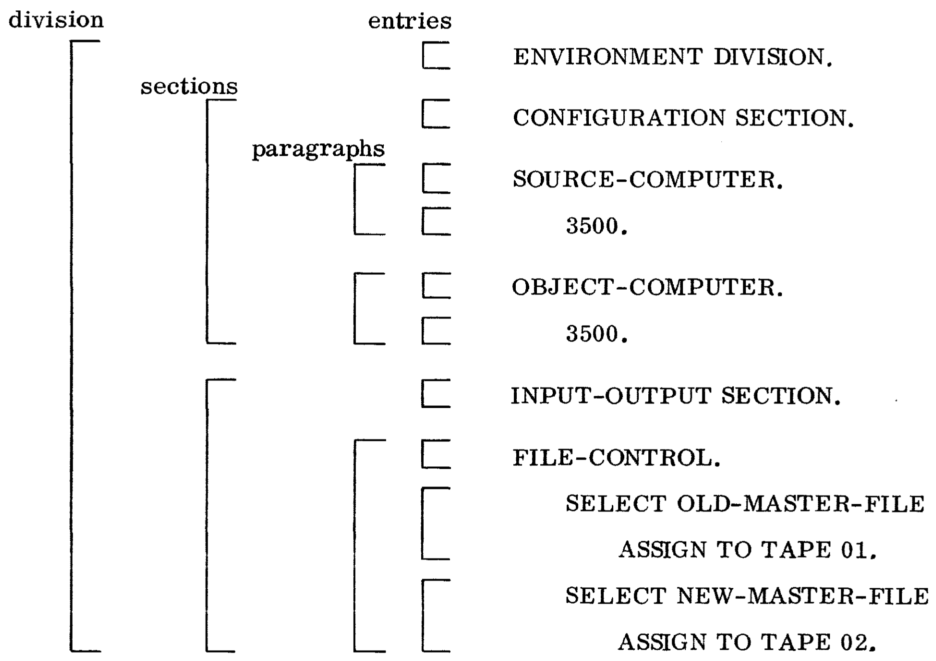
All divisions except the Data Division contain paragraphs. Paragraph names in the Identification and Environment Divisions are fixed. In the Procedure Division, paragraph names are user defined.

Paragraphs are identified by header entries consisting of a name followed by a period and a space. A paragraph header need not appear on a line by itself; it must be the first entry on a line, but may be followed on the same line by one or more entries in that paragraph.

Entries

An entry is one or a series of language elements terminated by a period and a space. The precise sequence of elements in each entry is dictated by the format rules contained in this manual.

Sample Division showing structural units



```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER.
    3500.

OBJECT-COMPUTER.
    3500.

INPUT-OUTPUT SECTION.

FILE-CONTROL.
    SELECT OLD-MASTER-FILE
        ASSIGN TO TAPE 01.
    SELECT NEW-MASTER-FILE
        ASSIGN TO TAPE 02.
```

PROGRAM CONTENTS

A COBOL source program must be identified, it must specify linkage between the computer system and the data files to be processed, it must define the data to be processed, and finally it must specify how the data is to be processed. A COBOL source program consists of the information required to perform these functions organized into the four required divisions.

Identification Division contains the information that identifies the program. At the very least, this division states the program name. In addition, it may contain the date the program was written, the date compiled, the author, the installation, and any remarks that might be useful to someone reading the program.

Environment Division contains information about the equipment used to compile and execute the program. Its primary function is to assign each data file by name to an input-output device and to specify any special input-output techniques.

Data Division contains a full description of the data to be processed by the object program. In the File Section, it describes the data items that make up each of the files named in the Environment Division. In the Working-Storage section it describes the data items used for constants and for data developed during execution of the program. If the program is divided into subprograms, the Common-Storage Section contains descriptions of data common to more than one subprogram. The Report Section contains a description of data to be output on a printed report in terms of the format in which it will be presented.

Data Division entries show how data items are grouped and organized into files and records. Data names, level numbers, pictures, and other information are contained in these entries.

Procedure Division specifies the actions required to process the data and it indicates the sequence of processing. The basic functions of this division are input-output, arithmetic, data movement, and sequence control. In addition, it provides the user with the capability to sort files, search tables, and write reports.

REFERENCE FORMATS

Each COBOL entry is described in terms of a reference format. Throughout this manual a format is shown adjacent to information defining the entry. When more than one specific arrangement is permitted, the format is separated into numbered formats.

Notation Used in Reference Formats

UPPER CASE	words are COBOL reserved words. They must be spelled correctly including any hyphens and may not be used in a source program except as specified in the reference formats.
UNDERLINED UPPER CASE	words are required when the format in which they appear is used. They define the action of the compiler.
lower-case-words	are generic terms which represent the words or symbols supplied by the user. When generic terms are repeated in a format, a number or letter is appended to the term for identification.
level numbers	are one- or two-digit numbers that precede entries in the Data Division to indicate the hierarchy level of entries.
[ ] Brackets	enclose optional portions of a reference format. All of the format within the brackets may be omitted or included at the user's option.

{ } Braces	enclose two or more vertically stacked items in a reference format when only one of the enclosed items can be used.
... Ellipses	immediately following a pair of brackets or braces indicate that the enclosed material can be repeated at the user's option.

Punctuation symbols (period, comma, semicolon) shown within the formats are required unless enclosed in brackets and specifically noted as optional.

Notation used in Examples

↑ indicates the position of an assumed decimal point in an item.

A plus or minus sign above a numeric character ($\overset{+}{n}$) indicates an operational sign is stored in combination with the numeric character.

Character positions in storage are shown by boxes, $\boxed{A}\boxed{B}\boxed{C}\boxed{D}$
An empty box means an unpredictable result.

△ indicates a space (blank).

CONTENTS

IDENTIFICATION DIVISON 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-paragraph.]
[INSTALLATION. comment-paragraph.]
[DATE-WRITTEN. comment-paragraph.]
[DATE-COMPILED. [comment-paragraph.]]
[SECURITY. comment-paragraph.]
[REMARKS. comment-paragraph.]
```

The header IDENTIFICATION DIVISION begins in column 8 of the first line and is followed by a period and a space. The name of each succeeding paragraph is specified on a new line; each begins in column 8 and is followed by a period and a space. The comment paragraphs are any combination of characters from the COBOL character set (appendix B) organized to conform to sentence and paragraph structure. Only the PROGRAM-ID paragraph is required.

1.1.1 PROGRAM-ID

The PROGRAM-ID paragraph gives the name by which the program is identified.

PROGRAM-ID. program-name.

The program name is a word and must conform to the rules for forming a word (appendix B). The program name identifies the source program, the object program, and all listings pertaining to the program. The first seven characters of this name must be unique within the COBOL source program.

1.1.2 DATE-COMPILED

DATE-COMPILED

The DATE-COMPILED paragraph provides the compilation date in the Identification Division source program listing.

DATE-COMPILED. [comment-paragraph.]

The paragraph name DATE-COMPILED causes the current date to replace the comment paragraph during program compilation.

The remaining paragraphs in the division supply documentary information and if specified, will appear in the source program listing.

ENVIRONMENT DIVISION 2

The Environment Division describes aspects of a COBOL program that depend on the physical characteristics of a specific computer. This division must be rewritten and the entire source program recompiled when the object program is to be run on different computers.

The Environment Division specifies the configuration of the compiling computer and the object computer. In addition, information relating to input-output control, special hardware characteristics, and control techniques can be specified. The information in the Environment Division is specified in two sections: The Configuration Section specifies the characteristics of the source computer and the object computer. The Input-Output Section specifies the information needed to control transmission and handling of data between external devices and the object program.

2.1 SPECIFICATION OF ENVIRONMENT DIVISION

```
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 Environment Division must begin with the header: ENVIRONMENT DIVISION followed by a period and a space.

2.2 CONFIGURATION SECTION

2.2.1 SOURCE-COMPUTER

This paragraph describes the computer upon which the program is to be compiled.

```
{ SOURCE-COMPUTER.  COPY statement. }
{ SOURCE-COMPUTER.  computer-name.  }
```

COPY is used only when the COBOL library contains the complete text of the SOURCE-COMPUTER paragraph. The COPY statement is described in section 6.1.

Computer name is a word which must appear as the first entry following SOURCE-COMPUTER. 3300 and 3500 are the only words permitted as computer names in this paragraph. The SOURCE-COMPUTER paragraph is used for documentation only, and has no effect on the object program.

2.2.2 OBJECT COMPUTER

This paragraph describes the computer on which the program is executed.

```
{ OBJECT-COMPUTER.  COPY statement.                                  }
{                                                                    }
{ OBJECT-COMPUTER.  computer-name                                    }
{                                                                    }
{   [                      { WORDS      } ]                          }
{   [ MEMORY SIZE integer  { CHARACTERS } ]                          }
{   [                      { MODULES    } ]                          }
```

COPY is used only when the COBOL library contains the complete OBJECT-COMPUTER paragraph. The COPY statement is described in section 6.1.

Computer name is a word and must appear as the first entry following the paragraph name OBJECT-COMPUTER. The computer names may be either 3300 or 3500. The integer memory size may be any value and may include alphabetic characters.

This paragraph is used for documentation purposes only and has no effect on the object program.

Example:

```
OBJECT-COMPUTER.  3500 MEMORY SIZE 262K WORDS.
```

2.2.3 SPECIAL NAMES

This paragraph assigns special control characters and implementor names to user defined mnemonic names. Two clauses permit the specification of non-standard currency conventions. If none of these features is required, this paragraph may be omitted.

Format 1:

```
SPECIAL-NAMES. COPY statement.
```

Format 1 is used only when the COBOL library contains the entire description of all SPECIAL-NAMES used in the program. The COPY statement is described in section 6.1.

Format 2:

```
SPECIAL-NAMES. [implementor-name IS mnemonic-name]...
               [,literal IS mnemonic-name]
               [,CURRENCY SIGN IS literal]
               [,DECIMAL-POINT IS COMMA].
```

Each mnemonic name defined under SPECIAL-NAMES must be unique within the source program.

implementor-name IS mnemonic-name

Implementor names are restricted to the following hardware names:

SYSTEM-INPUT	SYSTEM-PUNCH
SYSTEM-OUTPUT	CONSOLE

Mnemonic names must be used in references to system files or the console in an ACCEPT or DISPLAY statement in the Procedure Division.

literal IS mnemonic-name

The non-numeric literal is a one-character identifier used in either of the following:

carriage control in the WRITE ADVANCING mnemonic-name LINES statement

record prefix code in the CODE clause of the Report Description

CURRENCY SIGN IS literal

The non-numeric literal is used in the PICTURE clause to represent a currency symbol other than the $ (dollar sign). Alternate currency symbols are used for foreign currency. The literal must be a single character and may not be any of the following:

digits 0-9

alphabetic characters A B C D E P R S V X Z

space

special characters * + - , . ; ( ) "

If the CURRENCY SIGN clause is absent, only $ (dollar sign) may be used as a currency symbol in a PICTURE clause.

DECIMAL-POINT IS COMMA

This clause indicates that the function of comma and period are exchanged in any PICTURE clause character string and in numeric literals. This clause is useful for specifying foreign currency.

Example:

```
SPECIAL-NAMES.
     SYSTEM-OUTPUT IS PRINT,
     'A' IS EJECT, CURRENCY SIGN IS 'M',
     DECIMAL-POINT IS COMMA.
```

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 used, the entire section can be omitted.

2.3.1 FILE-CONTROL

This paragraph names each file, identifies the file device, and allows particular hardware assignment. It also specifies alternate input-output areas, the access mode and file limits of mass storage files, and the key to accessing random access files.

Format 1:

FILE-CONTROL. COPY statement.

Format 1 is used only when the COBOL library contains the entire text of the FILE-CONTROL paragraph. The COPY statement is described in section 6.1.

Format 2:

```
FILE-CONTROL.
                  { SELECT [OPTIONAL] file-name-1
                    [,RENAMING file-name-2]
                    ASSIGN TO [integer-1]   hardware-name-1 dsi-1
                                           [ { tui                  } ]
                                           [,{ hardware-name-2 dsi-2} ]
                   [,FOR MULTIPLE { REEL } ]
                   [              { UNIT } ]
                   [,RESERVE { integer-2 } ALTERNATE [ { AREA  } ] ]
                   [         { NO        }           [ { AREAS } ] ]
                   [,{ FILE-LIMIT  IS  } integer-3 [THRU integer-4] ]
                   [ { FILE-LIMITS ARE }                            ]
                   [,ACCESS MODE IS { RANDOM     } ]
                   [                { SEQUENTIAL } ]
                    [,PROCESSING MODE IS SEQUENTIAL]
                    [,ACTUAL KEY IS data-name-1]. } ...
```

If file-name-1 is a sort file, only the SELECT and ASSIGN clauses are included in the FILE-CONTROL paragraph. If file-name-1 is in the USING or GIVING options of the SORT verb, FOR MULTIPLE REEL/UNIT and RESERVE clauses may be included in addition to the required SELECT and ASSIGN clauses. (See SORT, section 4.7.24).

$\underline{\text{SELECT}}$ [$\underline{\text{OPTIONAL}}$] file-name-1

The SELECT clause is required for each file described in the Data Division. Each file is named only once as file-name-1 in the FILE-CONTROL paragraph following the key word SELECT. Each selected file must have a file description entry in the Data Division of the source program unless a RENAMING clause relates the selected file to a file that is the subject of another SELECT clause. The SELECT clause for the renamed file must not contain the RENAMING option.

The key word OPTIONAL is required for input files that are not necessarily present each time the object program is run.

$\underline{\text{RENAMING}}$ file-name-2

If the RENAMING clause is used, file-name-1 uses the file description entry, including the associated record description already specified for file-name-2. The files do not share the same area unless the SAME AREA clause appears in the I-O-CONTROL paragraph. File-name-1 is essentially a duplicate of file-name-2 except that file-name-1 has its own FILE-CONTROL paragraph entry. File-name-2, the renamed file, may not be a sort file.

$$\underline{\text{ASSIGN}}\ \text{TO [integer-1] hardware-name-1 dsi-1} \left[\left\{\begin{array}{l}\text{tui}\\ \text{hardware-name-2 dsi-2}\end{array}\right\}\right]$$

All files used in the program must be assigned to an input or output device (hardware name).

Allowable hardware names are as follows:

READER	SYSTEM-INPUT	TAPE	TTY (teletypewriter)
PRINTER	SYSTEM-OUTPUT	DISK	CRT (211 display)
PUNCH	SYSTEM-PUNCH	SCRATCH	

Each hardware name, except SYSTEM-INPUT, SYSTEM-OUTPUT and SYSTEM-PUNCH must be accompanied by a data set identifier (dsi). The dsi is a 1-4 character alphanumeric identifier used by MASTER and the input-output control system for all references to this file during the running of the program. It must not begin with an asterisk (*), and may not be INP, OUT, PUN or any COBOL reserved word.

Integer-1 indicates the number of input-output units of a given hardware name assigned to the file name. This option is used for documentation purposes only.

If alternating tape units are to be used, each must be given a unique dsi. Each file on a multi-file reel must have the same dsi. System files do not require dsi.

Examples:

```
    SELECT PAY-FILE ASSIGN TO TAPE A1, TAPE B2.
    SELECT PAY-FILE ASSIGN TO DISK MS01.
```

When a file is assigned to hardware names TTY and CRT, a second identifier code, terminal unit identifier, must follow the dsi. (When the hardware-name is TTY or CRT, only one hardware-name may be specified.) The terminal unit identifier (tui) is a 1-4 character alphanumeric code used by MASTER and the input-output control system to determine the local terminal unit associated with the assigned file. This identifier must match the code assigned to the device when it is entered into the MASTER operating system hardware unit table. For further information concerning the tui, refer to the MASTER 2.0 Reference Manual Pub. No. 60213600 and applicable installation procedures documentation.

Examples:

```
    SELECT TWX-FILE ASSIGN TO TTY ABS A001.
    SELECT DISPLAY-FILE ASSIGN TO CRT DEF B010.
```

All equipment assigned, except TTY and CRT, must be scheduled on the $SCHED card. Additionally, if the system printer, reader or punch are to be used directly by the program they must be preempted from MASTER with a $DIRECT card. (See Section 7, Source Program Preparation, Compilation and Execution).

SCRATCH files are automatically allocated by the I-O control system in the system scratch file area. These files exist only for the duration of the job and are released by MASTER at the end of each job. They may be processed and used in all respects as a sequential mass storage file might be used. Scratch area for these files must be scheduled with the $SCHED card (section 7.3.2).

If file-name-1 is a file, only the ASSIGN clause can follow the key word SELECT in the FILE-CONTROL paragraph (section 4.7.24, Sort).

FOR $\underline{\text{MULTIPLE}}$ $\left\{\begin{array}{l}\underline{\text{REEL}}\\ \underline{\text{UNIT}}\end{array}\right\}$

The MULTIPLE REEL clause must be included whenever the number of tape units assigned, explicitly or implicitly, might be less than the number of reels in the file. The MULTIPLE UNIT clause must be included whenever the number of mass storage devices assigned might be less than the number of mass storage units in the file. This can occur when the file is assigned to disk packs. If a random file is located on multiple disk packs, all such packs must be mounted on the drives at the time the file is opened.

$\underline{\text{RESERVE}}$ $\left\{\begin{array}{l}\text{integer-2}\\ \underline{\text{NO}}\end{array}\right\}$ ALTERNATE $\left[\left\{\begin{array}{l}\text{AREA}\\ \text{AREAS}\end{array}\right\}\right]$

The RESERVE clause allows the user to modify the number of input-output areas allocated by the compiler. The option RESERVE integer-2 ALTERNATE AREAS reserves integer-2 areas for the file in addition to the minimum area. Two additional alternate areas may be assigned for unblocked files, one for blocked files. The NO ALTERNATE AREAS option is documentary only. This clause does not apply to random files. Blocked random files have one alternate area; unblocked, no alternate areas. Alternate areas are never allocated to files assigned to TTY and CRT. All integers must be positive.

$\left\{\begin{array}{l}\underline{\text{FILE-LIMIT}}\ \underline{\text{IS}}\\ \underline{\text{FILE-LIMITS}}\ \underline{\text{ARE}}\end{array}\right\}$ integer-3 [$\underline{\text{THRU}}$ integer-4]

The FILE-LIMIT clause is applicable only to permanent mass storage files which will be internally allocated and/or expanded by the input-output control system at object time. Integer-3 indicates the number of blocks to be initially allocated to the file. The THRU integer-4 option applies only to sequential files.

During program execution the file limits logic operates as follows:

When an OPEN request is made on a mass storage file, the input-output control system attempts to open the file through the MASTER operating system. If the request to MASTER is rejected because the file does not exist, an internal allocation is performed, using the information furnished in the file description entry for the file. Open processing then continues in a normal manner.

If the allocated area is exceeded and the user has specified the THRU integer-4 option, the I-O control system automatically expands the file by ten percent of integer-3. This expansion may continue until the file reaches the maximum limit indicated by integer-4.

$\underline{\text{ACCESS}}$ MODE $\underline{\text{IS}}$ $\left\{\begin{array}{l}\underline{\text{RANDOM}}\\ \underline{\text{SEQUENTIAL}}\end{array}\right\}$

The ACCESS MODE clause must be included for mass storage files. When ACCESS MODE IS SEQUENTIAL, the mass storage records are obtained or placed sequentially. That is, the next logical record is made available from the file on execution of a READ statement, or a specific logical record is placed into the file on execution of a WRITE statement. No ACTUAL KEY entries are necessary for the SEQUENTIAL mode.

When ACCESS MODE IS RANDOM, mass storage records are organized randomly and records are retrieved according to the contents of the ACTUAL KEY data name.

$\underline{\text{PROCESSING}}$ MODE $\underline{\text{IS}}$ $\underline{\text{SEQUENTIAL}}$

This clause specifies that mass storage records are processed in the order in which they are accessed; it is documentary only.

$\underline{\text{ACTUAL}}$ KEY $\underline{\text{IS}}$ data-name-1

The ACTUAL KEY clause must be specified for random access files. The mass storage control system obtains or writes each record randomly. The specific logical record is located through data-name-1 of the ACTUAL KEY clause and is made available on execution of a READ statement. When a WRITE statement is executed for a random access file, effectively, the record is placed at the location in the file specified by data-name-1 of the ACTUAL KEY clause. Mass storage records in a random file are processed in the order in which they are accessed.

If a SEEK statement is executed prior to a READ or WRITE, the location of the record as specified by the value of data-name-1 is made available to the next READ or WRITE executed.

If no SEEK is issued, the record is automatically located from the value of data-name-1 upon execution of a READ or WRITE. The user is responsible for setting the value of data-name-1 prior to execution of each SEEK or READ or WRITE statement associated with a random access file.

Data-name-1 is a 12-character numeric group item, subdivided into two fields. The first is a seven-character block number indicating the relative position of this block from the beginning of the file (1 = the first block, etc.). The second field is a five-character position number giving the first character position of the record relative to the beginning of a block (0 = the first character, etc.).

The ACTUAL KEY item should be defined in the Data Division as follows:

```
01  RECORD-ADDRESS.
    02  BLOCK-NUMBER PIC 9(7).
    02  RECORD-POSITION PIC 9(5).
```

If data-name-1 points to the location of a record outside the specified file limits, the INVALID KEY clause of the READ or WRITE is executed.

The ACTUAL KEY clause is optional for sequential access files. If it is specified, the input-output control system updates data-name-1 by the following rules:

After a WRITE is executed, the contents of the ACTUAL KEY data item are always implicitly updated.

Before a READ is executed, the contents of the ACTUAL KEY data item are implicitly updated only if the READ was not logically preceded by a WRITE.

Since implicit updating of the ACTUAL KEY data item is a function of the input-output control system, changes made by the programmer to the ACTUAL KEY data item do not affect the mass storage file processing; but they may result in unpredictable values if the programmer makes subsequent reference to the contents of the ACTUAL KEY data item.

2.3.2 I-O-CONTROL

The I-O-CONTROL paragraph specifies the points at which rerun is to be established, the memory area to be shared by different files, and the location of files on a multiple-file reel.

Format 1:

<u>I-O-CONTROL.</u> <u>COPY</u> statement.

Format 1 is used only when the COBOL library contains the complete text of the I-O-CONTROL paragraph. The COPY statement is described in section 6.1.

Format 2:

```
    I-O-CONTROL.

     ┌ RERUN ON hardware-name-3 dsi-3                                  ┐
     │                ( END OF    ( REEL )  )                          │
     │       EVERY    <           ( UNIT )  >  OF file-name-3          │ ...
     └                ( integer-1   RECORDS )                          ┘
    ┌            ┌( SORT   )┐                                            ┐
    │; SAME      │<         >│  AREA FOR file-name-4 {, file-name-5} ... │ ..
    └            └( RECORD )┘                                            ┘
    [; MULTIPLE FILE TAPE CONTAINS file-name-6 [POSITION integer-3]
       [, file-name-7 [POSITION integer-4]] ... ] ... .
```

This paragraph is optional. The file name of a sort file cannot appear in a RERUN or MULTIPLE FILE option; nor can a sort file name appear in the SAME clause, unless the RECORD option is used. The END OF UNIT option may be used only if file-name-4 is a sequentially accessed mass storage file.

```
        RERUN ON hardware-name-3 dsi-3 EVERY
         ( END OF    ( REEL )  )
         <           ( UNIT )  >  OF file-name-3
         ( integer-1   RECORDS )
```

If the RERUN clause is specified, the contents of memory and of all necessary registers and the position of all files during execution are saved on the device specified by hardware-name-3. This device may be either magnetic tape or mass storage. If the dump file is to be on mass storage, the file must be allocated. In either case, it must be opened prior to execution with a *DEF card. The dsi parameter on the OPEN card must be identical to that specified in dsi-3.

The rerun dump can be used to restart execution at the point where a dump was taken (chapter 9, Source Program Execution). If the END OF REEL/UNIT option is used, a rerun dump is taken at the end of every reel or unit of the file named in file-name-3. The END OF UNIT condition occurs when the next sequential physical record is in a file segment of a disk pack other than the pack currently mounted on the device assigned to the file.

If the integer-1 RECORDS option is used, a rerun dump is taken each time the number of physical records specified by integer-1 is read from or written on the device associated with file-name-3.

$\underline{\text{SAME}}\ \left[\begin{Bmatrix}\underline{\text{SORT}}\\ \underline{\text{RECORD}}\end{Bmatrix}\right]\ \text{AREA FOR file-name-4}\ \left\{\text{, file-name-5}\right\}\ \ldots$

This clause specifies that two or more files are to use the same memory area during processing. If the RECORD option is specified, the files, including any sort-files, share only the area in which the current logical record is processed. Several files may be open at the same time but the logical record of only one file can reside in the record area at one time.

If the SORT option is used, one or more file names representing sort files may appear within the set of file names. For each sort file, the SAME SORT AREA clause is interpreted as follows:

> The SORT statement referring to the sort file and any associated input or output procedures can use all buffer storage associated with the other files in the SAME SORT AREA clause. All other files are thus guaranteed to be closed throughout operation of the SORT statement and any associated input or output procedures.
>
> Files other than sort files in the SAME SORT AREA clause do not share the same storage area. If other files are to share the same storage area, the user must write a SAME RECORD AREA clause.

If a SAME AREA clause does not contain the RECORD or SORT option, the shared area includes all storage areas assigned to the files; therefore, no more than one file may be open at one time.

$\underline{\text{MULTIPLE}}\ \underline{\text{FILE}}\ \text{TAPE CONTAINS file-name-6}\ [\underline{\text{POSITION}}\ \text{integer-3}]\ldots$

The MULTIPLE FILE clause is required when more than one file shares the same reel of tape. Regardless of the number of files on a single reel, only those files used in the object program need be specified. If all file names have been listed in consecutive order, the POSITION option need not be given. If any file in the sequence is not listed, the POSITION relative to the beginning of the tape must be given. Not more than one file on the same tape reel may be opened at one time.

Files on a multiple-file reel are handled in the same manner as other files except that OPEN and CLOSE perform the additional functions to locate each of the files on the reel.

The input-output control system searches for and locates any file on an input multifile reel which the user opens. If the indicated file is further along the tape than the current position, the tape is skipped forward; otherwise, the tape is rewound, then skipped forward to the correct file.

The files on an output multifile reel are written in the order dictated by the logic of the program rather than the order in which they are defined in the MULTIPLE FILE clause. If an output multifile reel is to be used as input to the same program, the input-output control system assumes that the files were written in the order specified by the MULTIPLE FILE clause.

The input-output control system assumes that all files on the multifile reel have the same type of labeling.

Since no two files on a multifile reel may be open simultaneously, they share buffer and record areas automatically. No file which shares a reel with another file can extend across reels; that is, a file cannot be on a multifile reel and also be multireel.

DATA DIVISION 3

The Data Division, required in every COBOL program, contains a full description of the data to be processed by the object program. The data falls into four categories:

Data stored on external files

Data common to more than one subprogram

Constants and data developed during program operation

Output reports

3.1 SECTIONS

The Data Division is divided into four sections corresponding to the categories of data:

```
DATA DIVISION.
[FILE SECTION.]
[COMMON-STORAGE SECTION.]
[WORKING-STORAGE SECTION.]
[REPORT SECTION.]
```

The Data Division must begin with the header: DATA DIVISION followed by a period and a space. Each of the sections is optional; when present, they must be in the order of appearance shown in the format above. Each section header is followed by one or more sets of entries; the File Section and Report Section contain file descriptions followed by associated record description entries, the Working-Storage and Common-Storage headers are followed by data description entries for independent items, followed by record descriptions.

3.1.1 FILE SECTION

The File Section defines the contents of data files stored on an external device and the contents of sort files. Each file is defined by a file description entry followed by one or more record description entries. A file description entry consists of a level indicator (FD or SD), a file name, and a series of independent clauses. An external file is described with a level FD entry which indicates recording mode, block size, labeling conventions, names of records or reports in the file, and so on. Every external file must have a level FD file description entry. Sort files are described with level SD entries which specify the name, size, and number of records to be sorted.

```
FILE SECTION.
FD file-name-1...
    01  data-name-1...
        02  data-name-2...
        66  data-name-3 RENAMES data-name-2.
                      .
                      .
                      .
            03  data-name-4...
    01  data-name-5...
        02  data-name-6...
        88  condition-name-1...
                      .
                      .
                      .
    01  data-name-7...
SD file-name-2...
    .
    .
    .
FD file-name-3...
    .
    .
    .
```

3.1.2 COMMON-STORAGE SECTION

During execution of an object program, independently compiled subprograms 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. All data structures permitted in the Working Storage Section are legal in the Common Storage Section.

A Common Storage Section must be present in each independently compiled subprogram so that all references to common storage will be properly defined. The item descriptions in each Common Storage Section should be identical in format; if they are not, the contents of common storage in memory cannot be guaranteed.

The system loader loads only the first Common Storage Section encountered. Subsequent Common Storage Sections are ignored and all reference to common storage items in communicating subprograms are directed to the first Common Storage Section loaded.

3.1.3 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.

The Working Storage Section is composed of the section header WORKING-STORAGE SECTION followed by a period. This header is followed by any independent data description entries which are in turn followed by any record description entries. Each name of an independent item or a record must be unique. Subordinate data names need not be unique if they can be made unique by qualification.

```
{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
                 .
                 .
                 .
```

3.1.4 REPORT SECTION

The Report Writer enables the user to specify the format of reports generated by a COBOL program. The Report Section is included when the Report Writer module is used. The content and format of reports are described in the Report Section. The header REPORT SECTION and a period is followed on succeeding lines by entries describing the physical format of each report to be produced.

Section 5 contains a complete description of the Report Writer and Report Section 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
          .
          .
          .
```

3.2 DATA DIVISION CONCEPTS

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. The Data Division contains three types of entry:

File description entries describe physical characteristics of a file.

Record description entries describe characteristics of items used in the program.

Report and report group description entries describe items generated as a report.

The Working Storage and Common Storage Sections contain record descriptions only; the File Section contains file descriptions and record descriptions; the Report Section contains report and report group descriptions.

3.2.2
ITEM

An item is an area used to contain data of a particular kind. The COBOL language recognizes four kinds of items:

elementary [illegible] not [illegible] [illegible]
group ~~[illegible]~~ subdivided
independent
report

Elementary items are items that are not subdivided into smaller items. A group item is subdivided into smaller items which can themselves be group or elementary items. Group and elementary items may be used in all sections of the Data Division. Independent items are unrelated elementary items not contained in a record; each is defined by the special level number 77. Independent items are illegal in the File Section. Report items are group or elementary items that appear in a report. They may be used only in the Report Section (see Report Writer, section 5).

3.2.3
RECORD

A record is the most inclusive item in the File, Working Storage or Common Storage Sections. It is usually, but not necessarily a group item. A record must have the level number 1 or 01 and a data name. Each item in a record must have at least a level number and a data name. In addition, any elementary items in a record must have at least a PICTURE or USAGE defined as INDEX, COMPUTATIONAL-1, or COMPUTATIONAL-2. Data names of items not at the 01 level in a record need not be unique if they can be made unique by qualification. In the File Section only, data names at the 01 level need not be unique since they can be qualified by a file name.

3.2.4
LEVEL NUMBER

The level number shows the hierarchy of data within a record description; the more inclusive an item, the lower its level number. Level numbers need not be consecutive as long as the less inclusive item has the higher number. The specific number is determined by the user. The first entry in a record description always has the level number 1 or 01. Group and elementary items within a record have level numbers in the range 2-49.

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

66 identifies RENAMES entries.

77 identifies independent items.

88 identifies condition-name entries.

3.2.5 DATA NAME

Every entry must have a subject; that is, a name assigned to the item. This data name is used to reference the item from the Procedure Division. It must not be qualified or subscripted when used as the subject of a Data Division entry.

The word FILLER may be used instead of a data name as the subject of an entry that is never referenced. It may name only an unused elementary item within a record. A record containing a FILLER item can be referenced from the Procedure Division, although the FILLER item itself cannot. A FILLER item must have a level number and any data description clauses that would be required for a data-name entry in the same location.

Report names, file names, library names, and condition names are special cases of the data name used in report, description, file description, source library, and condition name entries. Appendix B contains the rules for forming all data-names.

3.2.6 INITIAL VALUE

The initial value of any item in the Working Storage Section is specified with the VALUE clause in the record description entry.

3.2.7 LITERAL

A literal is an explicit statement of the value to be used in an operation performed by the object program. Literals may be used wherever the format indicates. Appendix B contains a description of literals and the rules governing their formation.

3.2.8 FIGURATIVE CONSTANT

Figurative constants are predefined as 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 B.

3.3 FILE DESCRIPTION ENTRY

A file description entry provides information about the physical structure, identification, and record names of each file used in the source program.

Format 1:

```
{FD}  file-name-1 COPY statement.
{SD}
```

Format 2:

```
   FD file-name-1

[                              ( {BINARY }    [ {LOW  }          ] )]
[                              ( {DECIMAL}    [ {HIGH } DENSITY  ] )]
[ ; RECORDING MODE IS          (              [ {HYPER}          ] )]
[                              ( {SECTOR}     [ {SEGMENTED } ]     )]
[                              ( {TRACK }     [ {CONTIGUOUS} ]     )]

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

[ ; RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS
      [DEPENDING ON {data-name-1 }]]
      [             {RECORD-MARK }]]

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

[ ; VALUE OF {label-field-1}  IS  {data-name-4}   [, {label-field-2}
[            {data-name-3  }      {literal-1  }   [  {data-name-5  }

IS {data-name-6}] ]
   {literal-2  }]...]

{ ; DATA {RECORD IS  }  data-name-7 [, data-name-8] ...   }
{        {RECORDS ARE}                                    }
{                                                         }
{ ; {REPORT IS  }  report-name-1 [, report-name-2] ...    }
{   {REPORTS ARE}                                         }

[; SEQUENCED ON data-name-9].
```

Format 3:

```
SD file-name-1
    [ ; RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS
    [
          [ DEPENDING ON { data-name-1   } ] ]
          [              { RECORD-MARK   } ] ]

      ; DATA { RECORD IS    }  data-name-2 [, data-name-3] ...
             { RECORDS ARE  }
```

Format 1 is used only when the COBOL library contains the entire FD or SD entry; format 2 describes external files; format 3 describes sort files. The COPY statement is defined in section 6.1.

The level indicator FD or SD identifies the beginning of each file description entry and must precede the file name. All semicolons are optional, but the entry must be terminated by a period. The clauses which follow the name of the file are optional in many cases, and their order of appearance is immaterial.

The REPORT clause of the FD entry is used only when Report Writer is used; it is described in chapter 5.

3.4 DATA DESCRIPTION ENTRY

A data description entry specifies the characteristics of each particular item of data.

Format 1:

$$01\ \text{data-name}\ \underline{\text{COPY}}\ \text{statement}$$

Format 2:

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

$$[;\ \underline{\text{REDEFINES}}\ \text{data-name-2}]$$

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

$$\left[;\ \text{USAGE IS}\ \left\{\begin{array}{l}\underline{\text{INDEX}}\\ \underline{\text{DISPLAY}}\\ \left\{\begin{array}{l}\underline{\text{COMPUTATIONAL}}\\ \underline{\text{COMP}}\end{array}\right\}\\ \left\{\begin{array}{l}\underline{\text{COMPUTATIONAL-1}}\\ \underline{\text{COMP-1}}\end{array}\right\}\\ \left\{\begin{array}{l}\underline{\text{COMPUTATIONAL-2}}\\ \underline{\text{COMP-2}}\end{array}\right\}\end{array}\right\}\right]$$

$$\left[;\ \underline{\text{OCCURS}}\ [\text{integer-1}\ \underline{\text{TO}}]\ \text{integer-2 TIMES}\ [\underline{\text{DEPENDING}}\ \text{ON data-name-1}]\right]$$

$$\left[\left\{\begin{array}{l}\underline{\text{ASCENDING}}\\ \underline{\text{DESCENDING}}\end{array}\right\}\ \text{KEY IS data-name-2 [, data-name-3]}\ \ldots\right]\ldots$$

$$[\underline{\text{INDEXED}}\ \text{BY index-name-1 [, index-name-2]}\ \ldots\ ]$$

$$\left[;\ \left\{\begin{array}{l}\underline{\text{SYNC}}\\ \underline{\text{SYNCHRONIZED}}\end{array}\right\}\quad \left\{\begin{array}{l}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{array}\right\}\right]$$

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

$$[;\ \underline{\text{BLANK}}\ \text{WHEN}\ \underline{\text{ZERO}}]$$

$$[;\ \underline{\text{VALUE}}\ \text{IS literal-3}].$$

Format 3:

$$66\ \text{data-name-1}\ \underline{\text{RENAMES}}\ \text{data-name-2}\ [\underline{\text{THRU}}\ \text{data-name-3}].$$

Format 4:

$$\text{88 condition-name ; } \left\{\begin{array}{l}\underline{\text{VALUE}}\ \text{IS}\\ \underline{\text{VALUES}}\ \text{ARE}\end{array}\right\} \text{ literal-1 [}\underline{\text{THRU}}\text{ literal-2]}$$

$$\text{[, literal-3 [}\underline{\text{THRU}}\text{ literal-4]] ... .}$$

Format 5:

$$\text{77 data-name-1} \left\{\begin{array}{l}\underline{\text{PIC}}\\ \underline{\text{PICTURE}}\end{array}\right\} \text{IS ... [USAGE IS]... [}\underline{\text{VALUE}}\text{ IS]... etc.}$$

All semicolons and commas are optional in the data description, but the entry must be terminated by a period.

Format 1 is used at the 01 level only. It may be used when the complete record description is contained in the COBOL library. The COPY statement is described in section 6.1.

Format 2 has a level number in the range 1-49. The clauses are written in any order with one exception: the REDEFINES clause, when used, must immediately follow the data name. PICTURE must be specified for every elementary item unless USAGE is INDEX, COMP-1, or COMP-2. The following clauses may be specified only at the elementary item level: PICTURE, SYNCHRONIZED, JUSTIFIED, and BLANK WHEN ZERO.

Format 3 is used with the RENAMES clause for alternate naming of elementary items or group items not at the 01 level. The level number must always be 66. The RENAMES clause must not be used with independent, level 77, entries.

Format 4 is used for condition name entries. Each condition name requires a separate entry with the level number 88. A condition name is a data name assigned to the condition; VALUE(S) specifies the value, values, or range of values associated with the condition name. A condition name entry must follow the entry describing the variable item with which it is associated. A condition name can be associated with any elementary or group item except the following:

> another condition name, a level 66 item, a group of items requiring special handling such as synchronization or usage, an index data item.

Format 5 describes independent items. An independent item is elementary, and is not part of a record or in the File Section. The level number must be 77. Independent items are described with the same clauses used in format 2 for elementary items.

3.5 DATA DIVISION CLAUSES

All clauses in the file and data description reference formats are described in this section with the exception of the REPORT clause which is described in section 5. The clauses are listed alphabetically.

3.5.1 BLANK WHEN ZERO

The BLANK WHEN ZERO clause permits the blanking of an item when its value is zero.

<u>BLANK</u> WHEN <u>ZERO</u>

When this clause is specified, the item will contain nothing but spaces if the value of the item is zero. This clause may be used only with a numerically edited item.

3.5.2 BLOCK CONTAINS

This clause specifies the size of a block or physical record.

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

BLOCK CONTAINS is required in a file description entry unless the block contains exactly one logical record. A file assigned to TTY or CRT cannot be blocked.

Integer-1 and integer-2 are positive integers.

For a mass storage file the size is stated in terms of RECORDS. The CHARACTERS option should be used if one of the following situations exists:

- Logical records extend across blocks
- The block contains padding, an area not contained in a logical record
- Logical records are grouped in such a manner that an inaccurate block size is implied

When the CHARACTERS option is used, the physical record size is specified in terms of the number of characters in the physical record. Characters are specified in standard data format regardless of their types.

If only integer-2 is shown, it represents the exact size of the physical record. If integer-1 and integer-2 are both shown, they refer to the minimum and maximum size of the physical record respectively.

If logical records of differing size are grouped into one physical record, the end of the logical record must be explicitly defined in the record description entry unless the RECORD CONTAINS DEPENDING ON option is used. The maximum file block size is 131,067 characters. Appendix F describes COBOL/MASTER file blocking formats.

Block

A block is a physical unit of data that is convenient for storage on an input or output device for a particular computer. It is identical with physical record. The block size has no direct relationship to the size of the file containing it, or to the size of logical records contained in the block. A block is normally composed of one or more logical records, or a portion of a logical record.

Logical Record

A COBOL logical record is a group of related information, uniquely identifiable, and treated as a unit. The concept of logical record is not restricted to file data but is carried over into the definition of working storage items. Logical records, whether in the File Section, Working Storage, or Common Storage Sections are defined by record description entries (section 3.2.3).

3.5.3 DATA RECORDS

The DATA RECORDS clause is required in each file description entry. The processor uses it to correlate file and record description entries.

$$\underline{\text{DATA}} \left\{ \begin{array}{l} \underline{\text{RECORDS}}\ \text{ARE} \\ \underline{\text{RECORD}}\ \text{IS} \end{array} \right\} \text{data-name-7 [, data-name-8] ...}$$

The data names are names of data records which must have 01 level numbers. The presence of more than one data name indicates that the file contains more than one type of data record. The order in which names are listed is immaterial.

Data records in a file need not have the same description. All data records within a file are processed from the same record area. The size of this area is equivalent to the largest record in the file.

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

This clause or the REPORT clause (Report Writer, section 5) must be included in each file description entry.

3.5.4 JUSTIFIED

The JUSTIFIED clause specifies nonstandard positioning of data within a receiving data item.

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

The JUSTIFIED clause can be specified only at the elementary item level in a record description entry.

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

Receiving Item	Rule for Positioning Data
Numeric	Data is aligned by decimal point and moved to the receiving character positions with zero fill or truncation on either end as required. If an assumed decimal point is not explicitly specified, the item is treated as if an assumed decimal point immediately followed the rightmost character.
Numeric Edited	Data is aligned by decimal point with zero fill or truncation at either end as required within the receiving data item, except where editing causes replacement of leading zeros.
Alphanumeric/ Alphabetic	Data is moved to the receiving character positions and aligned at the leftmost character position with space fill or truncation to the right.

The JUSTIFIED clause cannot be specified for an item described as numeric or for a numeric edited data item.

When the receiving item is described with the JUSTIFIED clause and the sending item is larger than the receiving data item, the leftmost characters are truncated; when the receiving item described with the JUSTIFIED clause is larger than the sending item, the data is aligned at the rightmost character position in the data item with space fill.

Example:

Picture	Data	Item	Justified	
S9(5)	$\boxed{1}\boxed{2}\boxed{\overset{+}{3}}$	$\boxed{0}\boxed{0}\boxed{1}\boxed{2}\boxed{\overset{+}{3}}\uparrow$	--	Right justified, zeros filled-in.
S9(4)V9	$\boxed{1}\uparrow\boxed{2}\boxed{\overset{+}{3}}$	$\boxed{0}\boxed{0}\boxed{0}\boxed{1}\uparrow\boxed{\overset{+}{2}}$	--	No justification, aligned by point.
S9(4)V9	$\boxed{1}\uparrow\boxed{2}\boxed{\overset{+}{3}}$	▨▨▨▨▨	Right	Illegal; item is numeric.
X(5)	$\boxed{A}\boxed{B}\boxed{C}$	$\boxed{\Delta}\boxed{\Delta}\boxed{A}\boxed{B}\boxed{C}$	Right	Right justified; blanks filled-in.
X(5)	$\boxed{A}\boxed{B}\boxed{C}$	$\boxed{A}\boxed{B}\boxed{C}\boxed{\Delta}\boxed{\Delta}$	--	Left justified normally.
X(2)	$\boxed{A}\boxed{B}\boxed{C}$	$\boxed{B}\boxed{C}$	Right	Right justified, left character truncated.

3.5.5
LABEL RECORDS

The LABEL RECORDS clause is required in every file description entry regardless of the presence or absence of label records. It identifies labels as standard or as nonstandard; but if labels are not present, it specifies that they are omitted.

Format 1:

$$\underline{\text{LABEL}} \left\{ \begin{array}{l} \underline{\text{RECORDS}}\ \text{ARE} \\ \underline{\text{RECORD}}\ \text{IS} \end{array} \right\} \underline{\text{STANDARD}}\ \underline{\text{VALUE}}\ \underline{\text{OF}}\ \text{label-field-1 IS}$$

$$\left\{ \begin{array}{l} \text{literal-1} \\ \text{data-name-1} \end{array} \right\} \left[\text{, label-field-2 IS} \left\{ \begin{array}{l} \text{literal-2} \\ \text{data-name-2} \end{array} \right\} \right] \ldots$$

Format 2:

$$\underline{\text{LABEL}} \left\{ \begin{array}{l} \underline{\text{RECORDS}}\ \text{ARE} \\ \underline{\text{RECORD}}\ \text{IS} \end{array} \right\} \text{data-name-3}\ \underline{\text{VALUE}}\ \underline{\text{OF}}\ \text{data-name-4 IS}$$

$$\left\{ \begin{array}{l} \text{literal-3} \\ \text{data-name-5} \end{array} \right\} \left[\text{, data-name-6 IS} \left\{ \begin{array}{l} \text{literal-4} \\ \text{data-name-7} \end{array} \right\} \right] \ldots$$

Format 3:

$$\underline{\text{LABEL}} \left\{ \begin{array}{l} \underline{\text{RECORDS}}\ \text{ARE} \\ \underline{\text{RECORD}}\ \text{IS} \end{array} \right\} \underline{\text{OMITTED}}$$

Format 1 is used when a file has standard labels (appendix E). All mass storage files except scratch files must be specified STANDARD in this clause. Mass storage scratch files, may have OMITTED or STANDARD labels. Mass storage file labels are maintained by MASTER File Supervisor and are not accessible to the user. For any non-mass storage file that has standard labels, the record area defined for that file is used by the input-output control system for processing the header and/or trailer labels. Thus, the user may access each label within the USE declarative procedures.

Format 2 is used when labels are nonstandard or user defined. Data-name-3 is the name of a label record area. It must not appear in the DATA RECORDS clause, and is either the subject of a record description associated with the file, or an area in working or common storage. Only the beginning labels are checked for files with data name labels. All Procedure Division references to the data name specified in this clause, or to any items subordinate to this data name, must appear within USE procedures.

Format 3 is specified when no explicit labels exist for the file or for the device to which the file is assigned. Files assigned to TTY or CRT do not have labels and OMITTED should be specified.

The VALUE OF clause is used only in formats 1 and 2. It specifies the values of data items used to identify or create the labels. A figurative constant may be substituted for any literal in the format.

The following label field names are used to define the values of standard labels:

Tape	Mass Storage
$\left\{\begin{array}{l}\text{ID}\\\text{IDENTIFICATION}\end{array}\right\}$	$\left\{\begin{array}{l}\text{ID}\\\text{IDENTIFICATION}\end{array}\right\}$
EDITION-NUMBER	$\left\{\begin{array}{l}\text{OWNER}\\\text{OWNER-ID}\end{array}\right\}$
REEL-NUMBER	EDITION-NUMBER
RETENTION-CYCLE	ACCESS-PRIVACY
	MODIFICATION-PRIVACY

The IDENTIFICATION or ID field is required. The specified label fields are checked against the label record by the input-output control system. If the physical file labels do not agree, the file is not processed. The values of these fields must exactly match the values given as parameters on any external file control cards. (*DEF cards, section 7.7.3).

When a label is nonstandard, the VALUE OF clause is required to define the values of identifying data items. Each of the subject data names (data-name-4, data-name-6, and so forth) is in the label record. The data names or literals following IS must be in the Working-Storage Section. The data names in this clause can be qualified but not subscripted or indexed, nor can they be described by USAGE IS INDEX.

VALUE OF causes the following action to be taken:

Input file	A label routine checks to see that the value of each label record data name or label field is equal to the value of its corresponding literal or data name. If any field does not compare, the file is rejected and a request is made to mount the correct file.
Output file	The value of each label record name or label field is made equal to the value of its corresponding literal or data name.

3.5.6 OCCURS

The OCCURS clause eliminates the need for separate entries when the items in a sequence are identical to each other in every respect except value. It also supplies information required to subscript or index the repeated items.

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

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

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

Each integer must be positive. If both are used, the value of integer-1 must be less than that of integer-2. The value of integer-1 can be zero; the value of integer-2 can never be zero.

The OCCURS clause is optional. It cannot be specified in a data description entry that has:

A level number of 01 or 77

Subordinate item containing the OCCURS...DEPENDING ON option

The OCCURS clause is used to define tables and other homogeneous sets of repeated data. The data name which is the subject of the data description entry containing the OCCURS clause must be either subscripted or indexed when it is referenced by any statement. If the entry containing OCCURS is a group item, each data name in the group must also be subscripted or indexed when referenced. The other clauses in an entry containing OCCURS apply to each occurrence of the item they describe. Three levels of nested OCCURS clauses are permitted.

The DEPENDING ON option is used to specify that the subject of the entry has a variable number of occurrences. Integer-1 represents the minimum number of occurrences and integer-2 the maximum. If the DEPENDING ON option is not specified, integer-2 is the exact number of occurrences and integer-1, if specified, is documentary only. Data-name-1 contains the count of the actual number of occurrences; its value must not exceed integer-2. Data-name-1 must be a positive COMPUTATIONAL integer; if it is described in the same record as the current entry, it must not be the subject of, or be subordinate to, an entry containing an OCCURS...DEPENDING ON clause. The value of data-name-1 should not be reduced below the actual number of occurrences of the data items; otherwise, the content of any data items whose occurrences exceed the new value of data-name-1 become unpredictable. An entry containing the DEPENDING ON option must not be the object of a REDEFINES clause.

When the DEPENDING ON option is specified in a data description in the File Section, records written on external devices will be variable in length depending on the value of data-name-1. DEPENDING ON may appear only once in a record description entry and must be at the last major level of the record.

The KEY IS option indicates that the repeated data is arranged in ascending or descending order according to the value of the key data names (data-name-2, data-name-3, etc.). This option is required when a SEARCH ALL statement is used in the Procedure Division to search the table of repeated data. The key data names are listed in descending order of significance. Data-name-2 either names the entry containing the OCCURS clause or names an entry subordinate to the entry containing OCCURS.

If data-name-3 or any succeeding key data names are used, each must name an entry subordinate to the group item which is the subject of the entry containing OCCURS. Except when data-name-2 is the subject of the OCCURS clause, none of the key data names can be the subject of, or subordinate to an entry containing an OCCURS clause.

The INDEXED BY option is required if the subject of this entry (or if it is a group, an item within it) is referred to by indexing or is referenced by the SEARCH statement. The index name is not defined elsewhere since its allocation and format is controlled by the compiler. It is not data, and cannot be associated with any data hierarchy. However, an index name must be initialized by the SET statement before it is used as a table reference. Index name items are one computer word in length and are in binary format (COMPUTATIONAL-1).

The VALUE clause must not be stated in a data description entry which contains OCCURS or is subordinate to an entry containing OCCURS. This rule does not apply to condition name entries. Initial values of items in working storage may be specified in the following manner:

> The table is described as a record with contiguous data description entries each of which has a VALUE clause as well as other clauses, (USAGE, PICTURE, etc.) required to complete the description. The table must be specified again with a REDEFINES clause to show its hierarchical structure. The entries subordinate to the REDEFINES entry are repeated because of the OCCURS clause; but they must not contain VALUE clauses.
>
> An alternate method can be used when the table elements do not require separate handling because of usage, synchronization, etc. In this case, the value of the entire table can be specified in the entry defining the table. The lower level entries specify the hierarchical structure of the table with OCCURS; they cannot contain VALUE clauses.

Examples:

```
1.  01  PARTS-LIST.
            02  A-PARTS.
                    03  PART-A1 PICTURE 9(10) VALUE xxxxxxxxxx.
                    03  PART-A2 PICTURE 9(10) VALUE xxxxxxxxxx.
                               .
                               .
                               .
                    03  PART-A100 PICTURE 9(10) VALUE xxxxxxxxxx.
            02  B-PARTS.
                    03  PART-B1 PICTURE 9(10) VALUE xxxxxxxxxx.
                    03  PART-B2 PICTURE 9(10) VALUE xxxxxxxxxx.
                               .
                               .
                               .
                    03  PART-B100 PICTURE 9(10) VALUE xxxxxxxxxx.
            02  C-PARTS.
                    03  PART-C1 PICTURE 9(10) VALUE xxxxxxxxxx.
                               .
                               .
                               .
            02  J-PARTS.
                    03  PART-J1 PICTURE 9(10) VALUE xxxxxxxxxx.
                               .
                               .
                               .
```

The above list contains a thousand entries, 100 lower level entries for each 02 level entry and 10 entries at the 02 level. To reference the list by subscripting or indexing, it must be redefined as:

```
01  PARTS-TABLE REDEFINES PARTS-LIST.
    02  LIST OCCURS 10 TIMES.
        03  PART PICTURE 9(10) OCCURS 100 TIMES.
```

2. If the list PARTS-LIST has a varying number of occurrences of the entries at the 02 level, it is redefined as follows:

```
01  PARTS-TABLE REDEFINES PARTS-LIST.
    02  PART PICTURE 9(10) OCCURS 1 TO 150 DEPENDING ON
        PART-NUMBER.
```

In this case, the number of occurrences is contained in the data item named PART-NUMBER with the following data description.

```
77  PART-NUMBER PICTURE 999 COMPUTATIONAL VALUE xxx.
```

Value must be a positive integer.

3. A table referenced by a SEARCH statement or by indexing can be described as follows:

```
01  DATA-LIST OCCURS 25 TIMES INDEXED BY LIST-INDEX,
    ITEM-INDEX.
    03  ITEMA PICTURE 9(3) OCCURS 10 TIMES
```

   The SET statement can be used to set the initial values of LIST-INDEX and ITEM-INDEX:

   SET LIST-INDEX TO 25. SET ITEM-INDEX TO 10.

4. A table to be searched by the SEARCH ALL statement can be described as follows:

```
01  DATA-LIST OCCURS 25 TIMES INDEXED BY LIST-INDEX,
    ITEM-INDEX DESCENDING KEY IS DATA-LIST, ITEM-KEY.
    03  ITEMA OCCURS 10 TIMES.
        05  AVALUE PICTURE 9(7)
        88  VALA VALUES ARE 1 THRU 999999.
    03  ITEM-KEY PICTURE X.
```

When a table area or record area requires subscripting or indexing, hardware indexing is employed only if the defined area is less than 16,383 characters. For a larger area, however, software logic is used for subscripting and indexing. Thus, subscripting and indexing may be slower if the defined area is greater than 16,383 characters.

3.5.7 PICTURE

The PICTURE clause describes general characteristics and editing requirements of an elementary item; it may be used only at the elementary item level.

$$\left\{\begin{array}{l}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{array}\right\} \text{ IS character string}$$

A character string consists of a maximum of 30 characters from the COBOL set. The particular combination of characters determines the category of the item. Six categories of data may be described with a PICTURE clause: alphabetic, numeric, alphanumeric, alphanumeric edited, numeric edited, and floating point edited.

The number of symbols that represent character positions indicate the size of an item. (Size means the number of character positions occupied by the item in standard data format.) For instance, an item containing 12 character positions is represented by a PICTURE clause containing either 12 symbols, AAAAAAAAAAAA, or the integer 12 in parentheses following the symbol, A(12). An integer in parentheses can be used to show the number of consecutive occurrences of the symbols: A X 9 P B 0 * , + - Z or the currency sign.

<u>Alphabetic</u>

The PICTURE string for an alphabetic item contains only the symbol A specified as many times as required. The item may contain any combination of letters of the alphabet and spaces not exceeding 131,067 alphabetic character positions.

<u>Numeric</u>

The PICTURE string for a numeric item is composed of the symbols 9 P S V. The item may contain any combination of numerals 0 1 2 3 4 5 6 7 8 9 and an optional plus (+) or minus (-) sign. The maximum size of the item is 18 numeric digit positions.

<u>Alphanumeric</u>

The PICTURE string for an alphanumeric item may contain the symbols A X and 9 only. It may contain all X's, but not all A's or all 9's; and an alphanumeric item is always treated as if it were all X's. Contents of the item are characters allowable in the computer's character set. The maximum size of the item is 131,067 alphanumeric character positions.

The following three categories are PICTURES of edited items. Editing alters the format or the punctuation of data in an item; characters can be suppressed or added.

Numeric Edit

The PICTURE string for a numeric edited item contains only the symbols B P V Z 0 9 , . / * + - CR DB and the currency sign. Allowable symbol combinations are determined by the order of precedence of symbols and editing rules. A maximum of 4095 character positions may be represented in the character string, but only 18 may be numeric digit positions. Contents of character positions representing digits must be numerals 0-9.

Alphanumeric Edit

The PICTURE string of an alphanumeric edited item contains the symbols A X 9 B 0; combinations are restricted to: at least one B and one X or at least one 0 and one X or at least one 0 and one A.

An alphanumeric edited item contains characters allowable in the computer's character set not exceeding 4095 alphanumeric character positions.

Floating Point Edit

The PICTURE string of a floating point edited item is restricted to the following symbols in the order listed:

+ or -

One to eleven 9's with leading, embedded, or trailing decimal point or scaling symbol V (the coefficient).

Letter E

+ or - (sign of the exponent)

Three 9's (value of the exponent)

The initial plus indicates that a plus sign represents positive values and a minus sign negative values; the initial minus indicates that blank represents positive values, a minus negative values. The maximum size is 18 alphanumeric character positions; the coefficient can have up to 11 numeric digit positions and the exponent 3 numeric digit positions.

CAUTION: The PICTURE clause defines the characteristics of the expected contents in a data item, and the code generated by the compiler is dependent upon the PICTURE descriptions. During object program execution, the actual contents of the data item are assumed to conform with the PICTURE. No data validation is ever performed by the generated code. If invalid data is introduced to the program the course of events is unpredictable.

Definition of Symbols Used in PICTURE

Symbol	Definition
A	Represents character position which can contain only a letter or a space; it is counted in size of item.
X	Represents a character position containing any allowable character from computer's character set; it is counted in size of item.
9	Represents a character position which contains a number; it is counted in the size of the item.
P	Indicates an assumed decimal scaling position when this position does not occur within the number that appears in the data item. It is not counted in the size of a numeric data item, but it is counted in determining the maximum size of a numeric edited item used as an operand in an arithmetic statement. P can appear only to the left or right in a string of one or more P's within the character string. The assumed decimal point is to the left of the leftmost P or to the right of the rightmost P; each P represents an implied character position (treated as zero) between the assumed decimal point and the leftmost character of the data item if the P's are to the left and the rightmost character if the P's are to the right. The assumed point location may be up to 31 places to the right or 30 places to the left of the rightmost digit position in the picture. The symbol V is redundant when P is specified. For example, a data item containing the number 2567 is treated as 256700 if the PICTURE is 9999PP; or as .002567 if the PICTURE is PP9999.
S	Indicates the presence of an operational sign (+ or -) and must be written as the leftmost character of a PICTURE string. It is not counted in determining the size of an item. (This character is used for documentation purposes only.)
V	Indicates the location of an assumed decimal point for aligning items during computation. It does not represent a character position and is not counted in determining the size of an item. V may appear only once in a PICTURE string. V is redundant when the assumed decimal point is to the right of the rightmost symbol in the PICTURE string.

Symbol	Definition
B	Represents a character position into which a space is inserted. It is counted in the size of the item.
Z	Represents the leftmost leading numeric character position to be replaced by a space when the contents of that position and all positions to the left of it are zero. Each Z is counted in the size of the item.
0 (zero)	Represents a character position into which the digit 0 will be inserted. Each 0 is counted in the size of the item.
, (comma)	Represents a character position into which the comma will be inserted. It is counted in the size of the item.
. (period)	Represents the decimal point for alignment purposes as well as a character position into which the period will be inserted. It is counted in the size of the item. For a given program the functions of the period and comma are exchanged if DECIMAL-POINT IS COMMA is specified in the SPECIAL-NAMES paragraph of the Environment Division. In this special case, all rules for a period apply to the comma, and rules for the comma apply to the period wherever they appear in a PICTURE clause.
+ - CR DB	Used as editing sign control symbols to represent the character position into which the symbol is placed. These four symbols are mutually exclusive in any one PICTURE string. Each character is counted in determining the size of the item.
* (asterisk)	Represents a leading numeric character position into which an asterisk is placed when the contents of that position is zero. Each * is counted in determining the size of the item.
/ (slash)	Represents a character position into which the slash character will be inserted. It is counted in the size of the item.
currency symbol $	Represents a character position into which the symbol will be placed. The dollar sign or a single character specified in the CURRENCY SIGN clause of the SPECIAL-NAMES paragraph in the Environment Division. It is counted in the size of the item.
E	Used in the PICTURE string of a floating point edited item to indicate the exponent. It is counted in the size of the item.

Examples:

PICTURE	Data Item	Result Item
Alphabetic		
AAAAA or A(5)	A B C D E	A B C D E
Numeric		
999	1 2 3	1 2 3 ↑
99V999	1 2 3 4 5	1 2 ↑ 3 4 5
S99V99	1 2 3 4	1 2 ↑ 3 $\overset{+}{4}$
PPP9999	1 2 3 4	↑0 0 0 1 2 3 4
SPPP9999	1 2 3 $\overline{4}$	↑0 0 0 1 2 3 $\overline{4}$
S999PPP	1 2 $\overline{3}$	1 2 $\overline{3}$ 0 0 0↑
Alphanumeric		
XXXXXXXX or X(8)	A B C D - * * *	A B C D - * * *
XXXXXXXX or X(8)	1 2 3 . 4 5 6 7	1 2 3 . 4 5 6 7
AAAX999	A B C / 1 2 3	A B C / 1 2 3

PICTURE	Data Item	Result Item
Numeric Edit		
$99	|4|8|	|$|4|8|
$$$99	|0|0|4|8|	|$|4|8|
ZZZ99	|0|0|9|2|3|	|Δ|Δ|9|2|3|
99.99	|4|8↑3|4|	|4|8|.|3|4|
*****.99	|0|0|0|1|0↑7|5|	|*|*|*|1|0|.|7|5|
++999	|0|0|2|8|2|	|+|2|8|2|
+999	|2|8|2|	|+|2|8|2|
Alphanumeric Edit		
BBBXXXB	|A|B|C|	|Δ|Δ|Δ|A|B|C|Δ|
000XXX	|1|2|3|	|0|0|0|1|2|3|
Floating Point Edit	**Value of Data**	
+9.E+999	+4.0	|+|4|.|E|+|0|0|0|
+999.9E+999	-3.125	|-|3|1|2|.|5|E|-|0|0|2|
+.99999E+999	+15275.	|+|.|1|5|2|7|5|E|+|0|0|5|
-999.99E+999	+74.325	|7|4|3|.|2|5|E|-|0|0|1|
-9.99E+999	-84.2	|-|8|.|4|2|E|+|0|0|1|

3.5.8 PICTURE EDIT

Editing Rules

Editing is accomplished by insertion or by suppression with replacement. Insertion editing consists of simple insertion, special insertion, fixed insertion, and floating insertion. Suppression with replacement editing consists of zero suppression with replacement by spaces or asterisks. Simple insertion editing is performed only on alphanumeric edited items. All types of editing may be performed on numeric edited items with the following restrictions:

> Floating insertion editing and suppression with replacement editing cannot be used in the same PICTURE.
>
> One type of replacement (either spaces or asterisks) with zero suppression can be used in the same PICTURE.
>
> A variable length item cannot be edited.

Simple Insertion Editing

Insertion characters are: , (comma), the letter B, / (slash), and 0 (zero). If the item is alphanumeric edited, only B or 0 may be used. Insertion characters are counted in the size of the item and represent the position in the item into which the character is inserted.

Examples:

PICTURE	Data Item	Result Item
9,999	0 1 5 4	0 , 1 5 4
99/99/99	0 1 2 3 6 7	0 1 / 2 3 / 6 7
09/99/99	1 2 3 6 7	0 1 / 2 3 / 6 7
BB99.99	0 1 5 7 ‸ 0 0	Δ Δ 5 7 . 0 0

Special Insertion Editing

The only special insertion character is the period. It also represents the decimal point for alignment purposes. When used as an actual decimal point, the period is counted in the size of the item. An assumed decimal point represented by the symbol V may not appear in the same PICTURE string as an actual decimal point represented by a period. When the period is the last symbol in the PICTURE string, it must be followed immediately by one of the punctuation characters, semicolon or period, followed by a space.

The result of special insertion editing is the appearance of the insertion character in the item in the same position as shown in the character string.

Fixed Insertion Editing

The fixed insertion characters are the currency symbol (cs) and the editing sign control symbols: + - CR and DB. Only one currency symbol and only one editing control symbol can be used in a given PICTURE string. CR and DB must be the rightmost character positions in the item; they are each counted as two character positions in determining the size of the item. The symbol + or - must be either the leftmost or the rightmost character position in the item. The currency symbol must be the leftmost character position unless it is preceded by a + or a -. The +, the -, and the currency symbol are each counted in determining the size of the item.

Fixed insertion editing results in the insertion of the editing symbol in the same position in the item as it occupied in the PICTURE. The result depends on the value of the item:

Symbol in Picture Character String	Result	
	Data Item Positive	Data Item Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

Examples:

PICTURE	Data Item	Result Item
+999	$\boxed{2}\boxed{9}\boxed{\bar{2}}$	$\boxed{-}\boxed{2}\boxed{9}\boxed{2}$
+999,999	$\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{\overset{+}{2}}$	$\boxed{+}\boxed{0}\boxed{0}\boxed{0}\boxed{,}\boxed{0}\boxed{1}\boxed{2}$
-999	$\boxed{2}\boxed{9}\boxed{\overset{+}{2}}$	$\boxed{\Delta}\boxed{2}\boxed{9}\boxed{2}$
-99.99	$\boxed{0}\boxed{2}\uparrow\boxed{9}\boxed{\bar{2}}$	$\boxed{-}\boxed{0}\boxed{2}\boxed{.}\boxed{9}\boxed{2}$
99.99DB	$\boxed{2}\boxed{3}\uparrow\boxed{7}\boxed{\bar{6}}$	$\boxed{2}\boxed{3}\boxed{.}\boxed{7}\boxed{6}\boxed{D}\boxed{B}$
99.99DB	$\boxed{2}\boxed{3}\uparrow\boxed{7}\boxed{6}$	$\boxed{2}\boxed{3}\boxed{.}\boxed{7}\boxed{6}\boxed{\Delta}\boxed{\Delta}$
99.99CR	$\boxed{1}\boxed{1}\uparrow\boxed{3}\boxed{\bar{4}}$	$\boxed{1}\boxed{1}\boxed{.}\boxed{3}\boxed{4}\boxed{C}\boxed{R}$
99.99CR	$\boxed{1}\boxed{1}\uparrow\boxed{3}\boxed{4}$	$\boxed{1}\boxed{1}\boxed{.}\boxed{3}\boxed{4}\boxed{\Delta}\boxed{\Delta}$
$BB999.99	$\boxed{2}\boxed{4}\boxed{3}\uparrow\boxed{2}\boxed{1}$	$\boxed{\$}\boxed{\Delta}\boxed{\Delta}\boxed{2}\boxed{4}\boxed{3}\boxed{.}\boxed{2}\boxed{1}$
$00999.99	$\boxed{2}\boxed{4}\boxed{3}\uparrow\boxed{2}\boxed{1}$	$\boxed{\$}\boxed{0}\boxed{0}\boxed{2}\boxed{4}\boxed{3}\boxed{.}\boxed{2}\boxed{1}$

Floating Insertion Editing

The symbols used for floating insertion are: the currency symbol (cs), and the editing sign symbols + or -. They are mutually exclusive as floating insertion characters in a given PICTURE string. Floating insertion editing is indicated by using at least two of the floating symbols in the leftmost numeric character positions. Any other insertion symbols embedded in the string of floating insertion symbols or to the immediate right of the string are part of the floating string.

Floating insertion in a PICTURE may be represented in two ways:

> Any or all leading numeric character positions to the left of the decimal point are insertion symbols.
>
> All numeric character positions in the picture are represented by the floating insertion symbol.

The first case results in a single insertion character placed in the position immediately preceding the first nonzero character or the decimal point, whichever is encountered first.

The result in the second case depends on the value of the data. If the value is zero, the entire edited item will contain spaces. If the value is not zero, the insertion character is placed in the position immediately preceding the first nonzero character or the decimal point, whichever is encountered first.

Examples:

PICTURE	Data Item	Result Item
------- or 7(-)	0 0 0 0 0 0	△ △ △ △ △ △ △
7(+)	0 0 0 1 2 3	△ △ △ + 1 2 3
4(+).99	0 0 0 1 ↑ 2 3	△ △ + 1 . 2 3
$$$$$.99	0 0 4 3 ↑ 5 9	△ $ 4 3 . 5 9
7($)	0 0 0 0 0 0	△ △ △ △ △ △ △
5($).99	0 0 0 0 ↑ 2 5	△ △ △ $. 2 5

The PICTURE character string must contain at least one more floating insertion character than the maximum number of significant digits in the item to be edited. All floating insertion characters are counted in the size of the item.

Zero Suppression Editing

Suppression of leading zeros in numeric character positions is indicated by the symbols Z or * in the PICTURE string. If Z is used, leading zeros are replaced by the space; if * is used, leading zeros are replaced by the asterisk. The PICTURE string contains one or more of either symbols in the leading numeric character positions. Any of the insertion characters embedded in or to the immediate right of this string are part of the string. The symbols + - * Z and the currency symbol (cs) are mutually exclusive within a given PICTURE string.

Zero suppression within a PICTURE string may be represented in two ways:

> Any or all leading numeric character positions to the left of the decimal point are represented by suppression symbols.
>
> All numeric character positions are represented by suppression symbols.

In the first case, any leading zero in the data which corresponds to a symbol in the string is replaced. Suppression terminates with the first nonzero digit or at the decimal point, whichever is encountered first.

In the second case, the result depends on the value of the data. If the value is zero, the entire data item is set to spaces when the suppression symbol is all Z or all *, except the decimal point when the symbol is *. If the value is not zero, the result is the same as if the suppression characters were only to the left of the decimal point.

When the asterisk is used as a suppression symbol and the clause BLANK WHEN ZERO also appears in the same entry, zero suppression editing overrides the function of BLANK WHEN ZERO.

Examples of Zero Suppression Editing:

PICTURE	Data Item	Result Item
ZZ999	0 0 9 2 3	△ △ 9 2 3
ZZZ,999.99	0 1 2 3 4 5↑	△ 1 2 , 3 4 5 . 0 0
Z99,999.99	0 0 1 2↑3 4	△ 0 0 , 0 1 2 . 3 4
$ZZZ,ZZ9.99	0 0 0 1↑2 3	$ △ △ △ △ △ △ 1 . 2 3
$ZZZ,ZZZ.99	0 0 0 0↑1 2	$ △ △ △ △ △ △ △ . 1 2
$***,**9.99	0 0 1 2 3 4↑	$ * * 1 , 2 3 4 . 0 0
$***,***.99	1 2 3 4 5 6↑	$ 1 2 3 , 4 5 6 . 0 0
$***,***.99	1↑2 3 4 5 6	$ * * * * * * 1 . 2 3
-ZZZ,ZZZ	0 0 0 0 1 $\bar{2}$↑	- △ △ △ △ △ 1 2
$ZZZ,ZZ9.99CR	1 2 3 4 5 $\bar{6}$↑	$ 1 2 3 , 4 5 6 . 0 0 C R
$ZZZ,ZZ9.99DB	0 0 0 1↑2 $\overset{+}{3}$	$ △ △ △ △ △ △ 1 . 2 3 △ △
$(4),$$9.99	0 0 1 2 3↑4	△ △ △ △ $ 1 2 3 . 4 0
$(4),$$$.99	0 0 0 0↑0 0	△ △ △ △ △ △ △ $. 0 0
ZZZZ.ZZ	0 0 0 0↑0 0	△ △ △ △ △ △ △
$$$$,$ZZ.99	0 0 0 0↑0 1	illegal picture

3.5.9 RECORD CONTAINS

The RECORD CONTAINS clause of a file description entry specifies the size of data records in the file.

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

$$\left[\underline{\text{DEPENDING}}\ \text{ON} \left\{ \begin{array}{l} \text{data-name-1} \\ \underline{\text{RECORD-MARK}} \end{array} \right\} \right]$$

This clause is not required. When present, however, the following rules apply:

> Integer-3 and integer-4 must be positive integers. Only integer-4 is used when all data records in the file have the same size. In this case, integer-4 represents the exact number of characters in each record. When integer-3 and integer-4 are both shown, integer-3 refers to the minimum number of characters in the smallest data record and integer-4 refers to the maximum number of characters in the largest data record.
>
> Size is specified in terms of the number of characters in the logical record. These characters are given in standard data format regardless of types within the logical record. Record sizes are determined according to the rules for obtaining the size of a group item. The maximum record size is 131,067 characters.

The DEPENDING ON option provides a key field (data-name-1) which governs the size of the record. Normally, it is a field within the record but may be located elsewhere. Data-name-1 may be a BCD integer or a binary number.

If data-name-1 is binary (COMPUTATIONAL-1) the key field is prefixed at the beginning of each record by the input-output control system. A prefixed key field format is called a universal record.

If RECORD MARK is specified, each record is terminated by a special character, external 32_8 (0,2,8 multiple punch). Reading or writing terminates when the RECORD MARK is encountered.

Files assigned to TTY or CRT must have fixed length logical records.

3.5.10 RECORDING MODE

The RECORDING MODE clause of a file description entry specifies the manner in which data is recorded on external storage devices.

$$\underline{\text{RECORDING}}\ \text{MODE IS} \left\{ \begin{array}{l} \left\{\begin{array}{l}\underline{\text{BINARY}}\\ \underline{\text{DECIMAL}}\end{array}\right\} \left[\left\{\begin{array}{l}\underline{\text{LOW}}\\ \underline{\text{HIGH}}\\ \underline{\text{HYPER}}\end{array}\right\} \quad \underline{\text{DENSITY}}\right] \\ \left\{\begin{array}{l}\underline{\text{SECTOR}}\\ \underline{\text{TRACK}}\end{array}\right\} \left[\left\{\begin{array}{l}\underline{\text{SEGMENTED}}\\ \underline{\text{CONTIGUOUS}}\end{array}\right\}\right] \end{array}\right\}$$

For magnetic tape, the density settings are: LOW=200 bpi, HIGH=556 bpi, and HYPER=800 bpi. If this clause is not present, a RECORDING MODE of DECIMAL HIGH DENSITY will be assumed.

The SECTOR/TRACK and SEGMENTED/CONTIGUOUS options apply only to mass storage files. When the FILE-LIMITS clause of the Environment Division is specified, the input-output control system will internally allocate and/or expand permanent mass storage files at object time. This allocation is performed according to the RECORDING MODE specification. SECTOR indicates that the minimum unit of allocation is a sector; TRACK that it is a track. SEGMENTED means that the file may be located in different areas of the same disk file. The MASTER file supervisor keeps track automatically of the locations of file portions. If CONTIGUOUS is specified, the file is assigned to one area large enough to hold the entire file on a single device (disk file or disk pack). The input-output control system automatically releases any unused portion of any file specified as SECTOR SEGMENTED. If RECORDING MODE is not specified for mass storage files, SECTOR SEGMENTED is assumed. Whenever possible, mass storage files should be allocated in CONTIGUOUS mode to minimize read/write head movement, thus increasing the rate of processing.

A file assigned to TTY or CRT is always assumed to have a recording mode of DECIMAL.

3.5.11 REDEFINES

The REDEFINES clause allows data items in the same physical area in memory to be specified in an alternate manner. The data is not changed, only the method of referencing the data; this includes giving the item a new name.

level-number data-name-1; <u>REDEFINES</u> data-name-2

The level number of data-name-1 must be the same as that specified for data-name-2; it may not be 66 or 88. This clause may not be at the 01 level in the File Section. Implicit redefinition is provided by the DATA RECORDS clause in the file description entry.

Entries giving the new description of the area in memory must immediately follow entries describing the area being redefined. Redefinition starts at data-name-2 and ends when a level number less than or equal to that of data-name-2 is encountered. Unless data-name-1 is at the 01 level, it must specify an area in memory the same size as that of data-name-2.

The data-name-2 entry cannot contain an OCCURS clause and cannot be subordinate to an entry containing an OCCURS clause. Nor can an OCCURS clause with the DEPENDING ON option be specified for any entries in the original item or its redefinition.

Example:

```
01  LIST.
    02  PART-1 PICTURE 999 VALUE IS xxx.
    02  PART-2 PICTURE 999 VALUE IS xxx.
    02  PART-3 PICTURE 999 VALUE IS xxx.
                      .
                      .
                      .
    02  PART-50 PICTURE 999 VALUE IS xxx.
01  NEW-LIST REDEFINES LIST.
    02  PARTS PICTURE 999 OCCURS 50 TIMES.
```

3.5.12 RENAMES

The RENAMES clause permits alternate, possibly overlapping, groupings of elementary items.

66 data-name-1, <u>RENAMES</u> data-name-2 [<u>THRU</u> data-name-3]

All RENAMES entries associated with a given entry must immediately follow the last data item description in the entry. One or more RENAMES entries may be written for a given record description.

Data-name-2 and data-name-3 must be names of elementary items or groups of elementary items in the associated record description entry; they must not be the same data name. Data-name-2 must precede data-name-3 in the record description. Data-name-3 must not be subordinate to data-name-2. A level 66 entry cannot rename another level 66 entry nor can it rename a level 77, level 88, or level 01 entry.

Data-name-1 must not be used as a qualifier and may be qualified only by the names of level 01 or FD entries. Data-name-2 and data-name-3 may be qualified. Neither data-name-2 nor data-name-3 may have an OCCURS clause in its data description entry nor be subordinate to an item that has an OCCURS clause in its data description entry.

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

When the THRU option is not specified, data-name-2 can be either a group or an elementary item. If data-name-2 is a group item, data-name-1 is treated as a group item; if data-name-2 is an elementary item, data-name-1 is treated as an elementary item.

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.
           66 PARTIAL-IDENT Renames VENDOR-Number
           thru CUST-IDENT
```

3.5.13
SEQUENCED ON

The SEQUENCED ON clause of a file description entry indicates the location of the identification field of a record accessed by RESPOND. RESPOND is the remote file processing system for the 3300/3500 computer configurations.

<u>SEQUENCED</u> ON data-name-9

If the FD entry for a file contains this clause and also specifies:

<u>RECORD</u> CONTAINS integer-1 TO integer-2 CHARACTERS
  <u>DEPENDING</u> ON data-name-n

where data-name-n is defined as

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

The input-output control system will insert the following information into the mass storage file label when the file is created:

Leftmost character position, relative to the beginning of the record, of the identification field (data-name-9)

Identification field length

Identification field mode (numeric or alphanumeric)

See appendix F for the description of the mass storage file label.

3.5.14 SYNCHRONIZED

The SYNCHRONIZED clause specifies positioning of an elementary item within a computer word or words.

$$\left\{\begin{array}{l}\underline{\text{SYNCHRONIZED}}\\ \underline{\text{SYNC}}\end{array}\right\} \quad \left\{\begin{array}{l}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{array}\right\}$$

This clause may appear only with an elementary item.

SYNCHRONIZED specifies that the COBOL compiler, in creating the internal format of this item, must place the item in the minimum number of computer words which can contain it. If the size of the item, explicit or implicit, is not an exact multiple of the number of characters in a computer word, the character positions between the item and the computer word boundary cannot be assigned to another item. Such unused character positions are included in:

the size of any group to which the elementary item belongs; and

the computer memory allocation when the elementary item appears as the object of a REDEFINES clause.

SYNCHRONIZED LEFT (SYNC LEFT) specifies that the elementary item is positioned to begin at the left boundary of a computer word.

SYNCHRONIZED RIGHT (SYNC RIGHT) specifies that the elementary item is positioned to terminate at the right boundary of a computer word.

When a SYNCHRONIZED item is referenced in the source program, the original size of the item, as shown in the PICTURE clause is used in determining any action that depends on size, such as justification, truncation, or overflow.

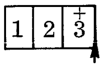
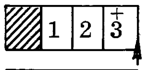
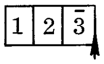
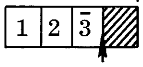
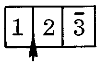
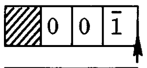
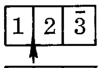
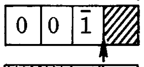
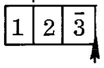
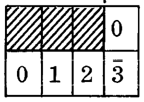
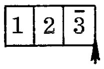
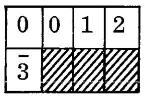
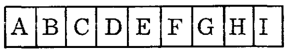
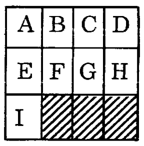
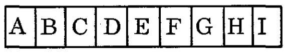
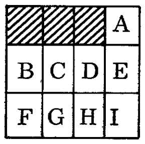
If the data description of an item contains the SYNCHRONIZED clause and an operational sign, the sign of the item appears in the normal operational sign position of the computer whether the item is SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT.

When the SYNCHRONIZED clause is specified for an item within the scope of an OCCURS clause, each occurrence of the item is SYNCHRONIZED. (See OCCURS.)

Items described as SYNCHRONIZED in the File Section are assumed on input to be synchronized on the external device, and on output are written in SYNCHRONIZED form on external device.

If the elementary item immediately preceding an item containing the SYNCHRONIZED clause does not terminate at a word boundary, the remaining character positions are regarded as FILLER and are not addressable.

Examples:

PICTURE	Data	SYNCHRONIZED	Result Item (machine words)
S9(3)V		RIGHT	
S9(3)V		LEFT	
S9(3)V		RIGHT	
S9(3)V		LEFT	
S9(5)		RIGHT	
S9(5)		LEFT	
X(9)		LEFT	
X(9)		RIGHT	

3.5.15
USAGE

The USAGE clause specifies the format of a data item in storage.

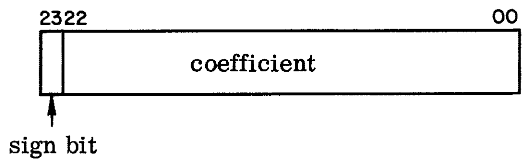
$$\text{USAGE IS} \left\{ \begin{array}{l} \underline{\text{DISPLAY}} \\ \left\{ \begin{array}{l} \underline{\text{COMPUTATIONAL}} \\ \underline{\text{COMP}} \end{array} \right\} \\ \left\{ \begin{array}{l} \underline{\text{COMPUTATIONAL-1}} \\ \underline{\text{COMP-1}} \end{array} \right\} \\ \left\{ \begin{array}{l} \underline{\text{COMPUTATIONAL-2}} \\ \underline{\text{COMP-2}} \end{array} \right\} \\ \underline{\text{INDEX}} \end{array} \right\}$$

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

If USAGE is not specified for an elementary item, or for any group to which the item belongs, usage is assumed to be DISPLAY. DISPLAY indicates that the data is standard data format. Standard data format uses the decimal system to represent numbers (regardless of the radix used by the computer), and the remaining characters in the COBOL character set to describe non-numeric data items.

A COMPUTATIONAL item contains a value to be used in computations; it must be numeric. The PICTURE string of a COMPUTATIONAL item without an integer suffix can contain only 9 S V and one or more P's. If a group item is described as COMPUTATIONAL, the elementary items in the group are COMPUTATIONAL; however, the group item, itself, is not COMPUTATIONAL since a group cannot be used in computations. A COMPUTATIONAL item is stored in standard data format and its sign is stored over the low-order numeric digit. Its size must not exceed 18 numeric digits.

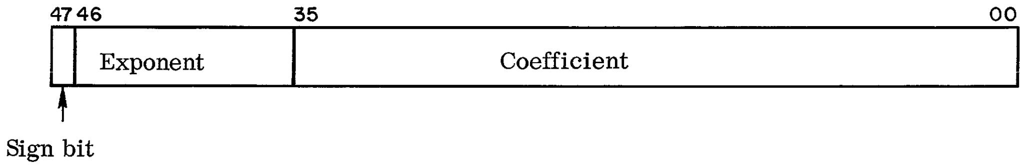
COMPUTATIONAL-1 describes an elementary item in fixed point binary format. Such an item occupies one computer word (24 bits) and its value may not exceed 2^{23}-1. A fixed point binary number consists of a sign bit and coefficient as shown below:



The binary point is assumed to be immediately to the right of the lowest order bit (00). The upper bit of any fixed number designates the sign of the coefficient (23 low order bits).

Sign Bit	Coefficient
1	Negative
0	Positive

COMPUTATIONAL-2 describes an item in floating point format. Such an item occupies two computer words (48 bits) and is stored as follows:



The coefficient consists of a 36-bit fraction in the low order positions of the floating point word. The coefficient is a normalized fraction; it is equal to or greater than 1/2 but is less than 1. The highest order bit position (47) is occupied by the sign bit of the coefficient.

Sign Bit	Coefficient
1	Negative
0	Positive

The floating point exponent is expressed as an 11-bit quantity with a value ranging from 0000 to 3777_8. It is formed by adding a true positive exponent and a bias of 2000_8 or a true negative exponent and a bias of 1777_8. This results in an effective exponent modulus of $\pm 1023_{10}$.

For further information concerning fixed point and floating point arithmetic, refer to the 3300 Computer Systems Reference Manual, Pub. No. 60157000.

The only meaningful clauses to be used with COMPUTATIONAL-n are the VALUE and REDEFINES clauses. COMPUTATIONAL-n items are synchronized left.

USAGE IS INDEX describes an elementary item as an index data item. An index data item is always one computer word in length in binary format; USAGE IS INDEX has the same effect as USAGE IS COMPUTATIONAL-1. An index data item is referred to by a SEARCH or a SET statement. It contains the character address bias of a table element. (The character address bias is the value added to the base character address of a table to give the character address of a particular element in the table.) An index data item can also appear in a relation condition, but it cannot be a conditional variable.

If a group is described with USAGE IS INDEX, all the elementary items in the group are assumed to be index data items. The group itself cannot be an index data item and cannot be referenced in a SEARCH or SET statement or a relation condition. The group can be referenced in a MOVE or an input-output statement; but no conversion takes place.

SYNCHRONIZED, JUSTIFIED, PICTURE, VALUE, and editing clauses cannot be used to describe group or elementary items for which usage is INDEX.

Examples:

1.
```
01 PRINT-LINE.
    02 FILLER PIC X(8) USAGE DISPLAY VALUE 'EMPLOYEE'.
    02 NAME PIC X(16) USAGE DISPLAY VALUE SPACES.
```

The described item would appear in memory as follows:

E	M	P	L
O	Y	E	E
Δ	Δ	Δ	Δ
Δ	Δ	Δ	Δ
Δ	Δ	Δ	Δ
Δ	Δ	Δ	Δ

2.
```
77 SUB-TOTAL PIC 9999V99 USAGE COMP VALUE ZEROS.
```

This item appears in memory as follows:

0	0	0	0	0	0

$\uparrow$ assumed decimal point

3.
```
01 OUTPUT-RECORD.
    02 RECORD-SIZE USAGE COMP-1 VALUE 120.
    02 FILLER PICTURE X(13) USAGE DISPLAY VALUE
       'MASTER-FILE-A'.
    02 FILE-DATE PICTURE X(8) VALUE SPACES.
```

This group appears in memory as follows:

00	00	01	70	binary value
M	A	S	T	
E	R	-	F	
I	L	E	-	
A	Δ	Δ	Δ	
Δ	Δ	Δ	Δ	
Δ	▨	▨	▨	

4. 77 TODAYS-DATE PIC X(8) USAGE DISPLAY VALUE '01/26/68'.
   77 SALARY-MAXIMUM PIC 9(4) V99 USAGE COMPUTATIONAL
      VALUE IS 5000.00.
   77 SALARY-TOTAL PIC 9(4) V99 USAGE IS COMP VALUE ZEROS.
   77 HOURS-WORKED USAGE COMP-1 VALUE ZEROS.
   77 MAX-HOURS-WORKED USAGE COMP-1 VALUE 60.
   77 COMP-VAL-1 USAGE COMPUTATIONAL-2 VALUE IS 3.46875E0.
   77 COMP-VAL-2 USAGE COMP-2 VALUE 12.279296E0.

The above items appear in memory as follows:

	0	1	/	2	TODAYS-DATE (8 BCD characters)
	6	/	6	8	
SALARY-MAXIMUM (6 BCD digits)	5	0	0	0	
	0	0	0	0	SALARY-TOTAL (6 BCD digits)
	0	0	0	0	
HOURS-WORKED (fixed point binary)	00	00	00	00	
	00	00	00	74	MAX-HOURS-WORKED (fixed point binary)
COMP-VAL-1 (floating point)	20	02	67	40	
	00	00	00	00	
	20	04	61	06	COMP-VAL-2 (floating point)
	00	00	00	00	
	▨	▨	▨	▨	
	▨	▨	▨	▨	

3.5.16 VALUE

The VALUE clause defines the initial value of working storage items or the values associated with a condition name.

Format 1:

$\underline{\text{VALUE}}$ **IS literal**

Format 2:

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

The VALUE clause cannot be stated for any item for which size is variable, explicitly or implicitly.

A figurative constant may be used wherever literal appears in the format.

If VALUE is specified for a group, the literal must be a figurative constant or a non-numeric literal, and the group area is initialized without regard for the individual elementary or group items. VALUE must not be specified for subordinate items within the group.

Format 2 is used only with condition names. In the THRU option, literal-1 must be less than literal-2, and literal-3 less than literal-4, etc. The VALUE clause is required in a condition name entry, and it is the only clause permitted in the entry. Characteristics of the condition name are implicitly those of its associated data item (its conditional variable).

The values specified in a VALUE clause must be consistent with any other clauses in the data description of the item. The following rules apply:

If the category of the item is numeric, all literals in the VALUE clause must be numeric literals. The literal is aligned according to the alignment rules given for the JUSTIFIED clause. The literal must not have a value requiring truncation of nonzero digits. Literals assigned to COMPUTATIONAL-2 items must be in the floating point literal format.

If the category of the item is alphabetic or alphanumeric, all literals in the VALUE clause must be non-numeric. The literal is aligned according to the rules; the number of characters in the literal must not exceed the size of the item.

All numeric literals in a VALUE clause must have a value within the range indicated by the PICTURE. For example, if the PICTURE is PPP99 the literal must be within the range .00000 through .00099.

The function of the editing clause or editing characters in a PICTURE is ignored in determining the initial appearance of the item. However, editing characters are included in determining the size of the item.

Rules governing the use of the VALUE clause differ in different sections of the Data Division:

Section	Rule
File Section	The VALUE clause may be used in condition name entries. It is documentary only in other entries.
Working-Storage/ Common-Storage Sections	The VALUE clause may be used in condition name entries and also to specify the initial value of any data item. The item assumes the specified value at the start of the object program. If VALUE is not specified, the initial value of an item is unpredictable.
Report Section	The VALUE clause causes the report data item to assume the specified value each time its report group is presented. This clause may be used only at the elementary level.

The VALUE clause must not be stated in a record description entry which contains an OCCURS clause or in an entry subordinate to an entry containing OCCURS. Nor can the VALUE clause be stated in an entry which contains a REDEFINES clause, or in an entry subordinate to an entry containing REDEFINES. These rules do not apply to condition name entries.

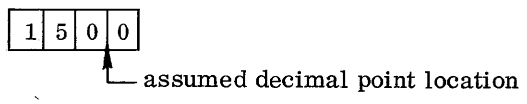
The VALUE clause must not be written for a group item containing descriptions that include SYNCHRONIZED or USAGE clauses.

Examples:

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

   77 FICA-MAX PICTURE 999V9 VALUE IS 150.0.

   The constant item will appear as follows:



2. Value is specified for the range 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. Execution of the object program begins with the first statement of the Procedure Division, excluding declaratives, and statements are executed in order of appearance until they are exhausted. The order of execution can be altered according to rules specified below.

Declaratives define procedures to be executed in addition to the standard error and label record handling procedures of the input-output control system. Procedures specified in a declarative are executed automatically under the input-output control system according to conditions specified in a USE statement. USE is the only declarative statement.

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.]
paragraph-name-2.
        .
        .
        .
paragraph-name-3.
[section-name-n SECTION.]
paragraph-name-n.
```

The division begins with the header and a period 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 contains the first section name followed by a space and the word SECTION. If sections are not specified, the next line is the first paragraph name followed by a period. Sections are optional, but if any paragraph in the division is in a section, then all paragraphs must be in sections.

Statements in the Procedure Division are combined to form sentences and paragraphs; paragraphs may be combined to form sections. Sentences are terminated by a period and a space; paragraphs by the next paragraph name, section name, or the end of the division. Sections are terminated by the next section name, the end of the division, or if they are declarative sections, by the terminator END DECLARATIVES. Paragraphs and sections are both called procedures; paragraph names and section names are referred to as procedure names. The elements of statements are COBOL words, identifiers, and literals. A summary description of these elements and of procedure names is contained in appendix B.

When the entire contents of a paragraph or section are contained in the COBOL library, the procedure name may be followed by the COPY statement. Section 6 contains a description of the COBOL library and the COPY statement.

4.1.1 DECLARATIVES

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. Each declarative is specified in a section by itself, preceded by a section header. The introductory sentence containing a USE statement follows the header. Associated procedures are specified according to the same rules as all other procedures in the program. The introductory USE sentence defines the conditions under which these procedures are to be executed by the input-output control system.

4.1.2 STATEMENTS AND SENTENCES

The three types of statements correspond to the three sentence types: imperative, conditional, and compiler directing. A semicolon, may be used as a separator between statements or within the IF statement to make a sentence more readable. A separator may not immediately follow another separator.

An imperative statement indicates a specific action to be taken by the object program. An imperative sentence is an imperative statement terminated by a period followed by a space. An imperative statement may consist of a sequence of imperative statements each separated optionally from the next by a separator. Imperative verbs are the following:

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

In a statement format, the term, imperative-statement, refers to a sequence of consecutive imperative statements ended by a period or an ELSE associated with a previous IF verb or a WHEN associated with a previous SEARCH verb.

A conditional statement specifies a condition to be evaluated for truth, and subsequent action of the object program is dependent on this truth value. A conditional sentence is a conditional statement or sequence of conditional statements optionally preceded by an imperative statement, terminated by a period followed by a space. Conditional statements are the following:

IF	RETURN
READ	WRITE with INVALID KEY
SEARCH	Arithmetic statements with ON SIZE ERROR

A compiler directing statement consists of a compiler directing verb and its operands. A compiler directing sentence is a single compiler directing statement terminated by a period followed by a space. Compiler directing verbs are the following:

COPY
ENTER
NOTE

† Without the SIZE ERROR option

†† Without the INVALID KEY option

4.2 ARITHMETIC EXPRESSIONS AND STATEMENTS

4.2.1 EXPRESSIONS

An arithmetic expression may be composed of the following elements:

Identifier of a numeric elementary item

Numeric literal

Identifiers and literals separated by arithmetic operators

Two arithmetic expressions separated by an arithmetic operator

Arithmetic expression enclosed in parentheses

Figurative constant: ZERO (S) (ES)

An identifier is a data name followed, as required, by the syntactically correct combination of qualifiers, subscripts, and indexes necessary to make unique reference to the data item. The identifiers and literals appearing in an arithmetic expression are numeric elementary items or numeric literals on which arithmetic may be performed. The USAGE of the identifiers is specified as:

COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2

The following arithmetic operators are used in arithmetic expressions:

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

Each operator is preceded by a space and followed by a space. Logical negation is expressed by a unary -.

An arithmetic expression may begin only with:

left parenthesis (unary -   variable

and may end only with:

right parenthesis)   variable

A one-to-one correspondence is necessary between left and right parentheses such that each left parenthesis is to the left of its corresponding right parenthesis.

Arithmetic expressions may be evaluated according to paired parentheses. Expressions within parentheses are evaluated first, and, within a test of parentheses, evaluation proceeds from the least inclusive to the most inclusive set. When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of operations is implied:

Unary -

**

* and /

+ and -

When the order of a sequence of consecutive operations is not on the same hierarchical level completely specified by parentheses, the order of evaluation is from left to right.

4.2.2 STATEMENTS

Arithmetic statements are ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT. They have several common features: data descriptions of the operands need not be the same; any necessary conversion and decimal point alignment is supplied throughout the calculation.

The maximum size of each operand is 18 decimal digits.

Operands may be COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2 items.

When the operands in arithmetic statements are not all the same mode, the compiler must generate conversions. In the following conversions:

COMPUTATIONAL-2 to COMPUTATIONAL-1

COMPUTATIONAL-2 to COMPUTATIONAL

COMPUTATIONAL-1 to COMPUTATIONAL

the converted value may not fit in the resultant field. The most significant digits could be lost making the result completely unreliable. To avoid this situation, the user should specify the ON SIZE ERROR option (see OPTIONS, 4.4.2). He may also use MOVE statements prior to the arithmetic statements to make the operands all the same mode.

4.3 CONDITIONS

A condition causes the object program to select between alternate paths of control depending on the truth value of a test. Conditions are used in IF, PERFORM, and SEARCH statements.

A condition is one of the following:

Relation condition

Class condition

Condition-name condition

Sign condition

NOT condition

Condition $\left\{ \begin{matrix} \underline{\text{AND}} \\ \underline{\text{OR}} \end{matrix} \right\}$

(Condition)

The construction: NOT condition, where condition is one of the first four types listed above, is not permitted if the condition itself contains NOT.

Conditions may be combined into logical operators. The logical operators must be preceded by a space and followed by a space.

Logical Operator	Meaning
OR	Logical Inclusive Or
AND	Logical Conjunction
NOT	Logical Negation

The figure below shows the relationships between the logical operators and conditions, A and B.

Condition		Condition and Value		
A	B	A AND B	A OR B	NOT A
true	true	true	true	false
false	true	false	true	true
true	false	false	true	false
false	false	false	false	true

4.3.1 RELATION CONDITION

A relation condition results in a comparison of two operands; the operands may be identifiers, literals, or arithmetic expressions. Comparison of two numeric operands is permitted regardless of the format specified by USAGE. For all other comparisons the operands must have the same usage specification.

General format for relation condition:

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

The first operand (to the left of the operator) is called the subject of the condition; the second operand (to the right of the operator) is called the object of the condition. The subject and object may not both be literals.

The relational operators specify the comparison to be made in a relation condition. They must be preceded by a space and followed by a space.

Relational Operator	Meaning
$\left\{\begin{array}{l}\text{IS [}\underline{\text{NOT}}\text{] }\underline{\text{GREATER}}\text{ THAN}\\ \text{IS [}\underline{\text{NOT}}\text{] }\underline{>}\end{array}\right\}$	Greater than, or not greater than
$\left\{\begin{array}{l}\text{IS [}\underline{\text{NOT}}\text{] }\underline{\text{LESS}}\text{ THAN}\\ \text{IS [}\underline{\text{NOT}}\text{] }\underline{<}\end{array}\right\}$	Less than or not less than
$\left\{\begin{array}{l}\text{IS [}\underline{\text{NOT}}\text{] }\underline{\text{EQUAL}}\text{ }\underline{\text{TO}}\\ \text{IS [}\underline{\text{NOT}}\text{] }\underline{=}\end{array}\right\}$	Equal to or not equal to

In any relation condition other than the first in a sentence, the subject, the subject and relational operator, or the subject and object may be omitted. The effect is the same as if the omitted parts were taken from the nearest preceding complete relation condition within the same sentence.

If in a consecutive sequence of relation conditions, separated by logical operators, the subjects are identical, the relational operators are identical, and the logical connectors are identical, the sequence may be abbreviated as follows:

Only the first occurrence of the subject and relational operator is written. The logical operator is written only once immediately preceding the last object.

```
IF X IS EQUAL TO 2 OR X EQUAL TO Y OR X EQUAL TO Z MOVE M TO N
```

can be abbreviated to

```
IF X IS EQUAL TO 2, Y, OR Z MOVE M TO N
```

Abbreviation 1: Identical subjects are omitted in a consecutive sequence of relation conditions.

```
IF A EQUAL TO B OR A IS LESS THAN C
```

can be abbreviated to

```
IF A EQUAL TO B OR IS LESS THAN C
```

and

```
IF A EQUAL TO B MOVE M TO N ELSE IF A IS LESS THAN C ADD
X TO Y
```

can be abbreviated to

```
IF A EQUAL TO B MOVE M TO N ELSE IF LESS THAN C ADD X TO Y
```

Abbreviation 2: Identical subjects and relational operators are omitted in a consecutive sequence of relation conditions.

```
IF A = B OR A = C
```

can be abbreviated to

```
IF A = B OR C
```

and

```
IF A = B ADD X TO Y ELSE IF A = C AND A = D MOVE M TO N
```

can be abbreviated to

```
IF A = B ADD X TO Y ELSE IF C AND D MOVE M TO N
```

Abbreviation 3: Identical subjects and objects are omitted in a consecutive sequence of relation conditions.

```
IF A EQUAL TO 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 EQUAL TO B OR IS GREATER MOVE C TO A IF GREATER
ADD B TO A
```

4.3.2 COMPARISON OF NUMERIC OPERANDS

A comparison of numeric operands results in the determination that the algebraic value of one is less than, equal to, or greater than the other. The length of the operands, in terms of number of digits, is not significant.

Zero is considered a unique value regardless of the sign. Comparison of these operands is permitted regardless of their usage.

4.3.3 COMPARISONS OF NONNUMERIC OPERANDS

A comparison between nonnumeric operands, or one numeric and one nonnumeric operand, results in the determination that one is less than, equal to, or greater than the other with respect to a specified collating sequence of characters.

The size of an operand is the total number of characters in the operand. Numeric and nonnumeric operands may be compared only when their usage is the same, implicitly or explicitly.

Operands of Equal Size

Characters in corresponding positions of the two operands are compared from the high-order end through the low-order end. If all pairs of characters compare equally, the operands are considered equal.

The first pair of unequal characters to be encountered is compared to determine their relative position in the collating sequence. The operand that contains the higher character in the collating sequence is considered to be the greater.

Operands of Unequal Size

The comparison of characters proceeds from high-order characters to low-order characters until a pair of unequal characters is encountered or until the characters in one of the operands are exhausted. If remaining characters in the longer operand consist entirely of spaces, the two operands are considered equal, otherwise the longer operand is considered greater.

4.3.4 COMPARISONS INVOLVING INDEX NAMES AND/OR INDEX DATA ITEMS

Comparison of two index names is the same as if the corresponding occurrence numbers are compared. Conversion from character address bias is performed automatically.

In the comparison of an index name with a literal or data item (other than an index data item), the occurrence number is compared to the actual value of the data item or literal. Conversion of the character address bias in the index name occurs prior to the comparison.

In the comparison of an index data item with an index name or another index data item, the actual values are compared without conversion.

The result of any other comparison involving an index data item is unpredictable.

4.3.5 SIGN CONDITION

This condition determines whether or not the algebraic value of a numeric operand is less than, greater than, or equal to zero.

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

An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero.

4.3.6 CLASS CONDITION

This condition determines whether the operand is numeric, that is, consists entirely of the characters 0, 1, 2, 3, ..., 9, with or without an operational sign, or whether it is alphabetic, that is, consists entirely of the characters A, B, C, ..., Z, space.

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

The operand being tested must be described, implicitly or explicitly, as USAGE DISPLAY.

The NUMERIC test cannot be used with an item described as alphabetic. The record description of the item being tested is determined to be numeric only if the contents are numeric.

The ALPHABETIC test cannot be used with an item described as numeric. The item being tested is determined to be alphabetic only if the contents consists of any combination of alphabetic characters and the space.

If the low-order character position of an otherwise numeric field contains a digit with a sign overpunch, the field is determined to be totally numeric. For a one-character alphanumeric field, that contains a numeric digit with a sign overpunch, both the NUMERIC and ALPHABETIC tests are considered true, and the results of the NOT option of the tests are false. An example of the latter case follows:

```
    .
    .
    .
02  FLD-A PIC X, VALUE 'A'
    .
    .
    .
IF FLD-A NUMERIC GO TO FLD-NUM
IF FLD-A ALPHABETIC GO TO FLD-ALPHA
IF FLD-A NOT NUMERIC GO TO FLD-ALPHA
IF FLD-A NOT ALPHABETIC GO TO FLD-NUM
```

The first two tests will take the true path; the last two tests will take the false path. Since 'A' in internal octal form is 21, it can be interpreted as either an alpha A or a numeric +1.

4.3.7 CONDITION-NAME CONDITION

A conditional variable is tested to determine whether or not its value is equal to one of the values associated with a condition name.

[NOT] condition-name

If the condition name is associated with a range or ranges of values, the conditional variable is tested to determine whether or not its value falls in this range, including the end values.

The rules for comparing a conditional variable with a condition name are the same as those specified for relation conditions.

The result of the test is true if one of the values corresponding to the condition name equals the value of its associated conditional variable.

4.3.8 EVALUATION RULES

Evaluation rules for conditions are analogous to those given for arithmetic expressions except that the following hierarchy applies:

arithmetic expression

all relational operators

NOT

AND

OR

4.4 COMMON OPTIONS IN STATEMENTS

Three options appear frequently in the statement descriptions that follow:

ROUNDED, SIZE ERROR, and CORRESPONDING

4.4.1 ROUNDED OPTION

Truncation occurs after decimal point alignment if the number of fractional places in the result of an arithmetic operation is greater than the number of fractional places provided for the result in the receiving item. Excess digits are truncated according to the format of the item containing the result, (result identifier). When rounding is requested, the absolute value of the result is increased by 1 when the most significant digit of the excess is greater than or equal to 5.

When the low-order integer positions in a result identifier are represented in a PICTURE by the character P, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

4.4.2 SIZE ERROR OPTION

A size error condition exists if the value of a result exceeds the largest value that can be contained in the associated result identifier after decimal point alignment. The size error condition applies only to final results, except in MULTIPLY and DIVIDE statements where size error applies to the intermediate results as well. Division by zero always causes a size error condition. If ROUNDED is specified, rounding takes place before checking for size error. When a size error condition occurs, subsequent action depends on whether or not the SIZE ERROR option is specified.

> If SIZE ERROR is not specified and a size error condition occurs, the value of the result identifier is unpredictable.
>
> If SIZE ERROR is specified and a size error condition occurs, the previous value of the result identifier is not altered. After the operation is completed, the imperative statement in the SIZE ERROR option is executed.

For CORRESPONDING arithmetic operations, the imperative statement in the SIZE ERROR clause is not executed until all individual additions or subtractions are completed.

Invalid non-numeric data contained within a numerically defined field does not constitute an ON SIZE error condition during arithmetic operations.

4.4.3 CORRESPONDING OPTION

For this discussion, d_1 and d_2 are each identifiers that refer to group items. A pair of data items, one from d_1 and one from d_2 correspond if the following conditions exist:

A data item in d_1 and a data item in d_2 have the same name and the same qualifications up to, but not including, d_1 and d_2.

At least one of the data items is an elementary data item in the case of a MOVE statement, and both of the data items are elementary numeric data items in the case of ADD or SUBTRACT.

Neither d_1 nor d_2 are data items with level number 66, 77, or 88.

When the CORRESPONDING option is specified, only the pairs of corresponding items as defined above are moved, added, or subtracted. When the groups identified by d_1 or d_2 contain items described with RENAMES, REDEFINES, or OCCURS clauses, those items are ignored; however, the groups themselves may be described with REDEFINES or OCCURS or be subordinate to items with these clauses.

4.5 TABLE HANDLING

The table handling functions provide a method for accessing tables of repetitive contiguous data items. Tables described with the OCCURS clause in the Data Division are scanned with the SEARCH statement in the Procedure Division. Items in tables also may be referenced by subscripting (up to three levels) or by indexing.

4.5.1 SUBSCRIPTS

Subscripts are used only when reference is made to an individual element in a list or table of like elements that have not been assigned individual data names. The subscript is a numeric literal integer, the special register TALLY, or a data name. The data name must identify a numeric elementary item that represents an integer. The data name used as a subscript may be qualified but not subscripted.

The subscript may contain a sign, but the lowest permissible subscript value is 1. The highest permissible subscript value, in any particular case, is the maximum occurrences of the item as specified in the OCCURS clause.

The subscript, or set of subscripts, that identifies the table element is enclosed in parentheses immediately following the terminal space of the table element data name. The table element data name appended with a subscript is called a subscripted data name or an identifier. When more than one subscript appears within a pair of parentheses, the subscripts must be separated by commas. A space must follow each comma, but no space may appear between the left parenthesis and the leftmost subscript or between the right parenthesis and the rightmost subscript.

$$\text{data-name} \left[\left\{ \begin{array}{c} \underline{\text{OF}} \\ \underline{\text{IN}} \end{array} \right\} \text{data-name-1} \right] \ldots \text{(subscript [, subscript] ...)}$$

4.5.2 INDEXING

Indexing is used to reference individual elements within a table of like elements. An index name is assigned to the level of the table in which it appears with the INDEXED BY option of the OCCURS clause when the table is described in the Data Division. The index name is initialized by a SET statement before it is used as a table reference.

Direct indexing is specified by using an index name as a subscript. Relative indexing is specified when the index name is followed by the operator + or - followed by an unsigned integral numeric literal all enclosed in the parentheses immediately following the terminal space of the data name.

Composite format:

$$\text{data-name}\left[\begin{Bmatrix}\text{OF}\\\text{IN}\end{Bmatrix}\text{data-name-1}\right](\text{index-name}\left[\begin{Bmatrix}+\\-\end{Bmatrix}\text{integer}\right]$$
$$\left[\text{, index-name}\left[\begin{Bmatrix}-\\+\end{Bmatrix}\text{integer}\right]\right]\ldots)$$

4.5.3
SEARCH FUNCTION

The search function operates in the linear mode (SEARCH) or the binary bisecting mode (SEARCH ALL). Both types of search terminate when a stated condition is met when table element equals search argument. All tables to be searched must have at least one element described as an index name with the INDEXED BY option of the OCCURS clause. The index name is incremented during search operations by the element character offset, and the index name always contains the character address bias of the table entry currently being compared. See paragraph 4.3.4 for rules governing the comparison of index names and index data items.

The element character offset is the number of character positions in a table element. The character address bias is the value added to the base character address of the table to give the character address of a particular element in the table.

4.5.4
RESTRICTIONS ON INDEXING, SUBSCRIPTING AND QUALIFICATION

Tables may have one, two, or three dimensions; therefore, references to an element in a table may require up to three subscripts or indexes.

A data name may not be subscripted nor indexed when the data name is used as an index, subscript, or qualifier.

When a data item requires qualification, subscripting or indexing, the indexes or subscripts are stated following all necessary qualification.

Subscripting and indexing must not be used together in a single reference.

Where subscripting is not permitted, indexing is also not permitted.

An index can be modified only by the SET, SEARCH, and PERFORM statements.

Index data items are described by USAGE IS INDEX. They permit storage of the values of index names as data with conversion.

The commas shown in the formats for indexes and subscripts are required.

4.6 DEBUGGING AID

The TRACE statement is a source language debugging aid. It is used to display the contents of data items and literals during job execution. The user can include any number of TRACE statements and he can specify the frequency and point of execution.

4.7 PROCEDURE DIVISION STATEMENTS

The following pages contain the Procedure Division statements in alphabetical order.

4.7.1 ACCEPT

The ACCEPT statement causes low volume data to be read from the system input file (INP) or the console.

ACCEPT identifier [FROM mnemonic-name]

The file or device must be associated with the mnemonic name in the SPECIAL-NAMES paragraph of the Environment Division.

ACCEPT causes the next set of data available at the file or device to replace the contents of the data item named by the identifier.

If the size of the data item is less than the fixed input unit (80 characters for system input, 127 characters for the console), the data appears as the first set of characters within the minimum unit. When data is input from INP, if the size of the data item is greater than 80 characters, multiples of 80 characters are read until the storage area allocated to the data item is filled. If the data item is greater than the fixed unit but is not an exact multiple, the remainder of the last fixed unit is not accessible.

If the FROM option is not given, data is accepted from the console.

4.7.2 ADD

The ADD statement causes two or more numeric operands to be summed and the result to be stored.

Format 1:

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

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

Values of operands are added together and the sum is stored in the location of the last operand specified. This operand may not be a literal.

Format 2:

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

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

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

Values of operands preceding the word TO are added together, then the sum is added to the current value in each identifier-m, identifier-n, ..., and the result is stored in each identifier-m, identifier-n, ..., respectively.

Format 3:

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

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

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

Values of operands preceding the word GIVING are added together, then the sum is stored as the new value of identifier-m, identifier-n, ...

Format 4:

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

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

Data items in identifier-1 are added to and stored in corresponding data items in identifier-2.

In Formats 1, 2, and 3 each identifier refers to an elementary numeric item; identifiers to the right of GIVING may refer to data items containing editing symbols. Each literal must be a numeric literal.

The composite of operands must not contain more than 18 digits. The composite is the data item resulting from superimposing all operands, aligned by decimal points. The data items that follow GIVING are not included in the composite. The compiler insures that enough places are carried to avoid loss of significant digits during execution.

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.

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             01  RATE-TABLE
    03  EASTERN-REG                   03  EASTERN-REG
        05  NEW-YORK                      05  NEW-YORK
            07  RATE                          07  RATE
        05  BOSTON                        05  BOSTON
            07  RATE                          07  RATE
    03  WESTERN-REG                       05  PHILADELPHIA
        05  LOS-ANGELES                       07  RATE
            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 UPDATE-RATE-TABLE are added to the rates for these three cities in the RATE-TABLE, and the results are the new rates. No other alteration occurs in the RATE-TABLE.

4.7.3 ALTER

The ALTER statement modifies a predetermined sequence of operations.

> <u>ALTER</u> procedure-name-1 <u>TO</u> [<u>PROCEED</u> <u>TO</u>] procedure-name-2
>
> [, procedure-name-3 <u>TO</u> [<u>PROCEED</u> <u>TO</u>] procedure-name-4] . . .

Each procedure-name-1, procedure-name-3, ... is the name of a paragraph that contains only one sentence consisting of a GO TO statement without the DEPENDING option.

Each procedure-name-2, procedure-name-4, ... is the name of a paragraph or section in the Procedure Division.

During execution of the object program, the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1, procedure-name-3, ... replacing the object of the GO TO by procedure-name-2, procedure-name-4, ..., respectively.

Example:

```
       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.

4.7.4 CLOSE

The CLOSE statement terminates the processing of input and output reels, units, and files with optional rewind and/or lock where applicable.

$$\underline{\text{CLOSE}}\ \text{file-name-1} \left[\begin{Bmatrix}\underline{\text{REEL}}\\ \underline{\text{UNIT}}\end{Bmatrix}\right] \left[\text{WITH} \begin{Bmatrix}\underline{\text{NO}}\ \underline{\text{REWIND}}\\ \underline{\text{LOCK}}\end{Bmatrix}\right]$$

$$\left[,\ \text{file-name-2} \left[\begin{Bmatrix}\underline{\text{REEL}}\\ \underline{\text{UNIT}}\end{Bmatrix}\right] \left[\text{WITH} \begin{Bmatrix}\underline{\text{NO}}\ \underline{\text{REWIND}}\\ \underline{\text{LOCK}}\end{Bmatrix}\right]\right] \ldots$$

Each file name is the name of a file upon which the CLOSE statement operates; it must not be the name of a sort file.

The REEL, LOCK, and NO REWIND options are applicable only to tape files. The UNIT option is applicable only to mass storage files in the sequential access mode. UNIT, throughout this specification means the storage device rather than the driver or power unit. Its use in the CLOSE statement is documentary and has the same effect as CLOSE file-name. It is illegal to specify CLOSE UNIT with the LOCK or NO REWIND options.

To explain CLOSE options for the various storage devices, all files are classified as follows:

Non-reel	A file on an input or output device for which concepts of rewinding and reels have no meaning (disk file, disk pack, drum, reader, printer, punch, TTY, CRT)
Single-reel	A file entirely contained on one reel of tape; the reel may contain more than one file
Multi-reel	A file contained on more than one reel of tape

A CLOSE statement with no options performs the following standard close operations on non-reel, single-reel, or multi-reel files:

Input files:	If the file is positioned at its end and there is an ending label record, the ending label record is checked and the data area is released. If the file is positioned at its end and there is no ending label record, or if the file is not positioned at its end, the data area is released but no ending label checking takes place. An input file is positioned at its end when the AT END imperative statement has been executed but no CLOSE statement has yet been executed.
Output files:	If an ending label record has been described for the file, it is constructed and written on the output device and the data area is released.
Input-output files:	Regardless of the position of the file, the data area is released.

Single or multi-reel files are rewound; the file is positioned at its beginning on the last (or only) used reel. The previous reels are not affected.

If a mass storage file is described with a FILE LIMITS clause, the input-output control system automatically releases the unused portion of the area allocated to the file when the file is closed. For a mass storage file CLOSE UNIT has the same effect as CLOSE FILE.

CLOSE WITH LOCK applies only to tape files. It performs the standard close operations for single-reel and multi-reel files, rewinds the file, and unloads it. The file is no longer available for processing.

CLOSE WITH NO REWIND applies only to tape files. The file is closed with the standard close procedures but the current reel remains in its current position.

When multi-reel files are closed, previous reels are not affected unless controlled by a prior CLOSE REEL. If the current reel is not the last in an input file, succeeding reels are not processed in any way. CLOSE REEL applies only to multi-reel tape files. The following operations are performed in a standard close reel:

Input files:	Reel swap and standard beginning reel label and user's beginning reel label procedures (if specified by USE). Order of execution is specified in USE statement.
	Makes available the next data record on the new reel.
Output files:	Standard ending reel label and user's ending reel procedure (if specified by USE). Order of execution is specified in USE statement.
	Reel swap and standard beginning reel label procedure and user's beginning reel label procedure if specified by USE.

The file is rewound, positioned at its beginning on the last used reel.

CLOSE REEL WITH LOCK causes the standard close reel operations to be performed on the current reel of a multi-reel tape file. In addition, the reel is rewound and unloaded. The reel cannot be processed again as a part of the file.

CLOSE REEL WITH NO REWIND causes the standard close reel operations to be performed on the current reel of a multi-reel tape file but the reel is left in its current position.

4.7.5 COMPUTE

The COMPUTE statement assigns to data items the value of a numeric data item, literal, or arithmetic expression.

$$\underline{\text{COMPUTE}}\ \text{identifier-1}\ [\underline{\text{ROUNDED}}]\ ,\ [\text{identifier-2}\ \ \underline{\text{ROUNDED}}]]\ \ldots$$

$$\left\{\begin{array}{l}\underline{\text{FROM}}\\ \underline{=}\\ \underline{\text{EQUALS}}\end{array}\right\}\ \left\{\begin{array}{l}\text{identifier-n}\\ \text{literal-1}\\ \text{arithmetic-expression}\end{array}\right\}$$

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

The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on the composite of operands and/or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE.

Each identifier must refer to an elementary numeric item, except that identifiers to the left of the equals sign may describe data items containing editing symbols. Literal-1 must be a numeric literal. The arithmetic expression option permits the use of any meaningful combination of identifiers, numeric literals, and arithmetic operators, parenthesized as required. The words FROM and EQUALS are equivalent to the equals sign (=).

The maximum size of each operand is 18 decimal digits.

The identifier-n and literal-1 options provide a method for setting the values of identifier-1, equal to the value of identifier-n or literal-1.

Examples:

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

COMPUTE DATA-1 = 100.
```

4.7.6 DISPLAY

The DISPLAY statement causes low volume data to be written on the system output file (OUT), the system punch file (PUN), or the console.

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

The mnemonic name is associated with a system file or the console in the SPECIAL-NAMES paragraph in the Environment Division. Each literal may be any figurative constant except ALL.

DISPLAY causes the contents of each operand to be written on the system file or the console in the order listed. If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.

When a DISPLAY statement contains more than one operand, the data comprising the first operand is stored as the first set of characters, the data comprising the second operand as the second set of characters, and so on, until the maximum output file and 127 characters for the console) is filled. This operation continues until all information is displayed. Data comprising an operand may extend into subsequent units.

If the UPON option is not used, the data is displayed on the console.

COMP-2 items are converted to a standard 18-character format: +.9(11)E+999 before being displayed.

COMP-1 items are converted to a standard 8-character format: +9(7) before being displayed.

All other items are displayed without conversion.

4.7.7 DIVIDE

The DIVIDE statement divides one numeric data item into another and sets the value of a data item equal to the results.

Format 1:

$$\underline{\text{DIVIDE}} \left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\end{array}\right\} \underline{\text{INTO}} \text{ identifier-m } [\,\underline{\text{ROUNDED}}]$$

$$[\text{, identifier-n } \underline{\text{ROUNDED}}]] \ldots [\text{; ON } \underline{\text{SIZE}} \ \underline{\text{ERROR}} \text{ imperative-statement}]$$

The value of identifier-1 or literal-1 is divided into the initial values of identifier-m, identifier-n,... Dividend values are replaced by the quotient values.

Format 2:

$$\underline{\text{DIVIDE}} \left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\end{array}\right\} \underline{\text{INTO}} \left\{\begin{array}{l}\text{identifier-2}\\ \text{literal-2}\end{array}\right\} \underline{\text{GIVING}} \text{ identifier-m}$$

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

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

The initial value of identifier-1 or literal-1 is divided into identifier-2 or literal-2 and the result is stored in identifier-m, identifier-n,... respectively.

Format 3:

$$\underline{\text{DIVIDE}} \left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\end{array}\right\} \underline{\text{BY}} \left\{\begin{array}{l}\text{identifier-2}\\ \text{literal-2}\end{array}\right\} \underline{\text{GIVING}} \text{ identifier-m}$$

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

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

The value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2 and the result is stored in identifier-m, identifier-n,... respectively.

Each identifier must refer to a numeric elementary item, except that any identifiers to the right of the word GIVING can refer to data items that contain editing symbols. Each literal must be a numeric literal.

The maximum size of each operand is 18 decimal digits.

Examples:

1. DIVIDE 1.8 INTO CONVERTED-TEMP1 CONVERTED-TEMP2 CONVERTED-TEMP3 CONVERTED-TEMP4.

2. DIVIDE ABLE INTO BAKER GIVING CHARLEY ROUNDED ON SIZE ERROR GO TO RECOMPUTE.

3. DIVIDE TOTAL BY DAILY GIVING RATE ROUNDED RATE-A ROUNDED ON SIZE ERROR PERFORM DIVIDE-FAULT-ROUTINE.

4.7.8 ENTER

The ENTER statement makes it possible to use more than one language in the same program.

```
ENTER language-name; routine-name [; (parameter-string)]
```

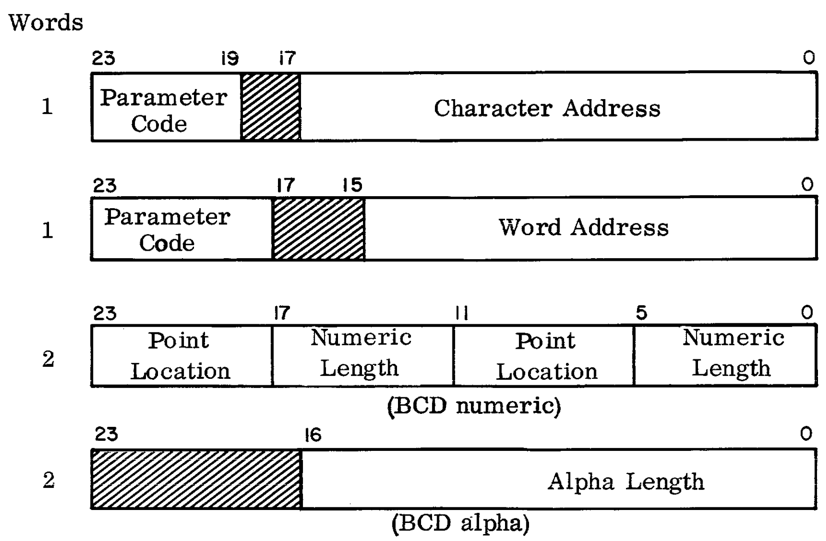
Language-name may be COBOL, FORTRAN, or COMPASS for USASI COBOL, USASI FORTRAN, COMPASS/MASTER. Statements in the other language are executed as if they had been compiled into the object program following the ENTER statement.

Routine name identifies the portion of coding to be executed at this point in the procedure sequence. A routine name is a COBOL word which may be referred to only in an ENTER sentence. It is 1-8 characters and must conform to requirements for an entry point name recognizable by the MASTER loader. It is automatically declared as an EXTERNAL symbol by the USASI COBOL compiler.

Parameter strings may contain any or all of the following: procedure names, file names, identifiers, non-numeric literals, and signed numeric literals.

Parameter names must be separated by commas. A numeric literal must be prefixed by plus or minus to distinguish it from an all numeric procedure name.

Parameters occupy 24, 36, or 48 bits in storage. The compiler analyzes the parameter name list and codes and structures the parameters as shown below. A procedure name or a file name and floating point or binary integer parameters require one word only; BCD parameters require two words.



Code	Type	Address	Chapter	Descriptive Word
00	Procedure name	word		
40	File name	word		
52	BCD Alpha	character	2	follows
56	BCD Alpha	character	1	follows
60	BCD Numeric	character	2	follows
61	BCD Numeric	character	2	precedes
64	BCD Numeric	character	1	follows
65	BCD Numeric	character	1	precedes
70	Binary Interger	word	2	
71	Floating Point	word	2	
74	Binary Integer	word	1	
75	Floating Point	word	1	

The descriptions of BCD numeric items are in the half-word preceding or following the parameter depending on the code. A one- or two-chapter method of addressing is indicated by the code (see MASTER Reference Manual for description of chapters). The word address of a file name is the beginning address of the File Environment Table entry associated with that file name.

The size and address of a literal are passed as parameters in the same manner as the size and address of identifiers. When a subscripted identifier is listed in the ENTER parameter string, the compiler computes the actual subscripted address and passes it in the appropriate parameter word.

Before control is passed to the routine named by ENTER, the mode of program execution is automatically relocated to instruction state and, upon return from the routine, is relocated to operand state.

Refer to Appendix J for ENTER Verb examples.

4.7.9 EXAMINE

The EXAMINE statement replaces or counts the number of occurrences of a given character in a data item.

```
            / TALLYING   { UNTIL FIRST  }  literal-1 [REPLACING  \
            |            { ALL          }    BY literal-2]       |
EXAMINE     |            { LEADING      }                        |
identifier  <                                                    >
            |            { ALL          }                        |
            | REPLACING  { LEADING      }  literal-3 BY literal-4 |
            \            { [UNTIL] FIRST }                       /
```

The description of the identifier must be USAGE is DISPLAY (explicitly or implicitly).

Each literal must consist of a single character belonging to a class consistent with that of identifier. A literal may be any figurative constant, except ALL.

Examination proceeds as follows:

- Non-numeric data items are examined from left to right. Each character in the data item specified by the identifier is examined in turn.

- Numeric data item referred to by EXAMINE must consist of numeric characters. Examination starts at the leftmost character and proceeds to the right. Each character except the sign is examined in turn. The sign is ignored by EXAMINE.

The TALLYING option creates an integral count which replaces the value of a special register called TALLY (Special Register, appendix B). The count represents the number of:

- Occurrences of literal-1 when the ALL option is used.

- Occurrence of literal-1 prior to a character other than literal-1 when the LEADING option is used.

- Characters not equal to literal-1 before the first occurrence of literal-1 when the UNTIL FIRST option is used.

For the REPLACING options, the replacement rules are:

- When the ALL option is used, literal-2 or literal-4 is substituted for each occurrence of literal-1 or literal-3.
- When the LEADING option is used, the substitution of literal-2 or literal-4 terminates as soon as a character other than literal-1 or literal-3 or the righthand boundary of the data item is encountered.
- When the UNTIL FIRST option is used, the substitution of literal-2 or literal-4 terminates as soon as literal-1 or literal-3 or the righthand boundary of the data item is encountered.
- When the FIRST option is used, the first occurrence of literal-1 or literal-3 is replaced by literal-2 or literal-4.

4.7.10 EXIT

The EXIT statement provides a common end point for a series of procedures.

<u>EXIT</u>.

The word EXIT appears in a sentence by itself. It must be preceded by a paragraph name and be the only sentence in the paragraph.

It is sometimes necessary to transfer control to the end point of a series of procedures. Normally control is transferred to the next paragraph or section, but in some cases this does not have the required effect. For instance, the point to which control is to be transferred may be at the end of a range of procedures governed by a PERFORM or at the end of a declarative section. The EXIT statement is provided to enable a procedure name to be associated with such a point.

If control reaches an EXIT paragraph and no associated PERFORM or USE statement is active, control passes through the EXIT point to the first sentence of the next paragraph.

4.7.11 GO TO

The GO TO statement causes control to be transferred from one part of the Procedure Division to another.

Format 1:

```
GO TO [procedure-name-1]
```

Format 2:

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

Each procedure name is the name of a paragraph or section in the Procedure Division. Identifier is the name of a numeric elementary item with no positions to the right of the assumed decimal point.

Whenever a format 1 GO TO statement is executed, control is transferred to procedure-name-1 or to another procedure name if the GO TO statement has been altered by an ALTER statement.

If procedure-name-1 is not specified in format 1, an ALTER statement referring to this GO TO statement must be executed prior to the GO TO statement. When the GO TO statement is referred to by an ALTER statement the following rules apply:

- GO TO statement must have a paragraph name.
- GO TO statement must be the only statement in the paragraph.

If a format 1 GO TO statement appears in an imperative sentence, it must be the only or last in a sequence of imperative statements.

A format 2 GO TO statement causes control to be transferred to procedure-name-1, procedure-name-2, ..., procedure-name-n, depending on whether the value of the identifier is 1, 2, ..., n. If the value of identifier is other than a positive or unsigned integer in the range 1 to n, control passes to the next statement.

Examples:

1. ADD 1 TO COUNT.
2. IF COUNT GREATER 3 GO TO CONTINUE.
3. GO TO PR1 PR2 PR3 DEPENDING ON COUNT.

4.7.12 IF

The IF statement causes a condition to be evaluated; subsequent action of the object program depends on whether the value is true or false.

$$\underline{\text{IF}}\ \text{condition;}\ \left\{\begin{array}{l}\text{statement-1}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{array}\right\}\left[\ ;\ \underline{\text{ELSE}}\ \left\{\begin{array}{l}\text{statement-2}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{array}\right\}\right]$$

Statement-1 and statement-2 represent a conditional statement or an imperative statement; either may be followed by a conditional statement.

The phrase ELSE NEXT SENTENCE may be omitted if it immediately precedes the terminal period of the sentence.

When an IF statement is executed, the following action is taken:

- If the condition is true, statements immediately following the condition are executed; control then passes implicitly to the next sentence.
- If the condition is false, either statements following ELSE (statement-2) are executed, or if the ELSE clause is omitted, the next sentence is executed.

If the words NEXT SENTENCE are written and the condition is met, control passes explicitly to the next sentence.

Examples:

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

2.  IF A IS NUMERIC MULTIPLY A BY B; 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.
                      .
                      .
                      .
```

If statement-1 and statement-2 contain an IF statement, the IF statement is said to be nested.

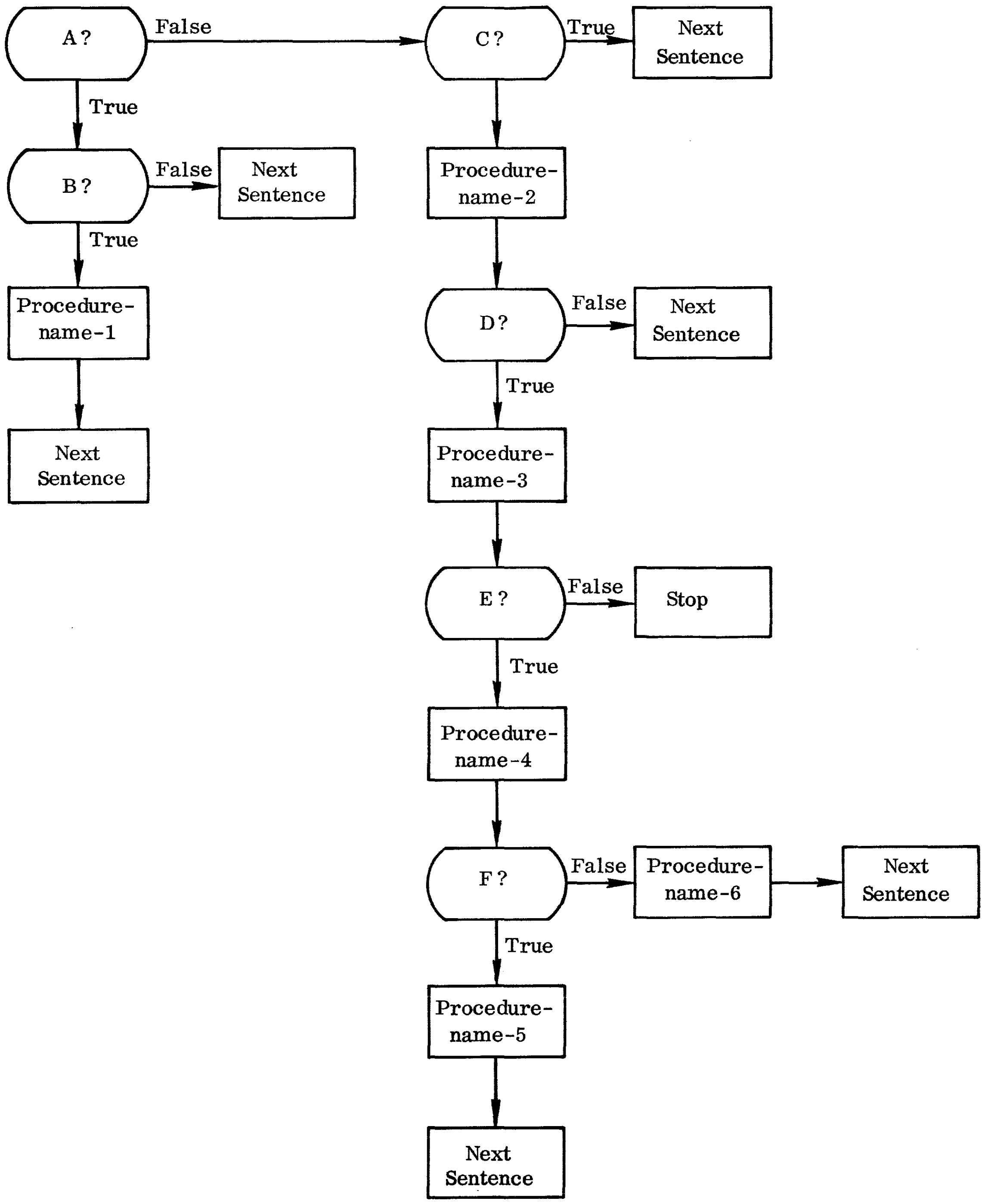
IF statements within IF statements may be considered as paired IF and ELSE combinations, proceeding from left to right. Any ELSE encountered is considered to apply to the immediately preceding IF that has not been already paired with an ELSE.

When control is transferred to the next sentence, implicitly or explicitly, control passes to the next sentence as written or to a return mechanism of a PERFORM or a USE statement.

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; 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.



4.7.13 MOVE

The MOVE statement transfers data, in accordance with the rules of editing, to one or more data areas.

Format 1:

$$\underline{\text{MOVE}} \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \underline{\text{TO}} \text{ identifier-m [, identifier-n] . . .}$$

Format 2:

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

In format 1, identifier-1 and literal-1 represent the sending area; identifier-m, identifier-n. . . , represent the receiving area.

The data designated by the literal or identifier-1 is moved first to identifier-m, then to identifier-n, etc.

When both the sending and receiving items are elementary the move is elementary. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, alphanumeric edited, floating point edited. Numeric literals belong to the category numeric, and nonnumeric literals belong to the category alphanumeric.

The following elementary MOVES are illegal:

<u>Source Item Category</u>	<u>Receiving Item Category</u>
Numeric Edited Alphanumeric Edited Floating Point Edited Alphabetic	Numeric Numeric Edited
Numeric Numeric Edited Floating Point Edited	Alphabetic
Numeric (non-integer)	Alphanumeric Alphanumeric Edited

All other elementary MOVES are legal, and are performed according to the following rules:

MOVE to Alphanumeric Edited, Alphanumeric, Alphabetic:

Justification and any necessary space-filling takes place as defined under the JUSTIFIED clause. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated after the receiving item is filled.

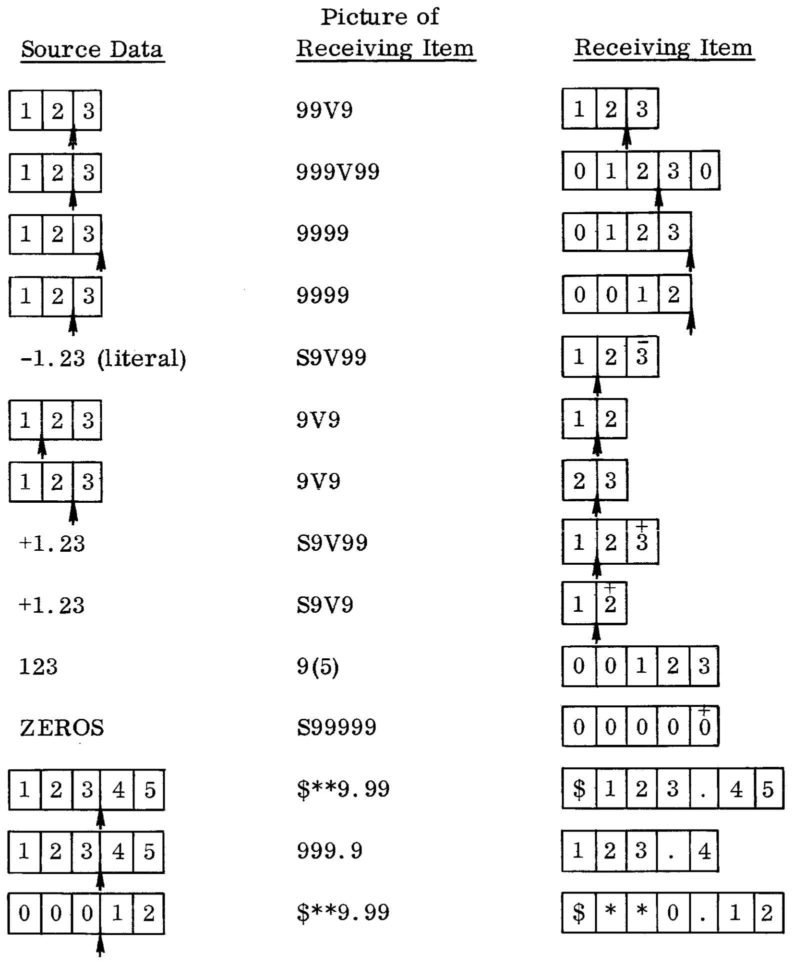
Examples:

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
"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)

MOVE to Numeric, Numeric Edited, Floating Point Edited:

Alignment by decimal point and any necessary zero-filling is done automatically except where zeros are replaced because of editing requirements. If the sending item has more digits to the left or right of the decimal point than the receiving item can contain, excess digits are truncated. If the sending item contains any non-numeric characters, the results at object time are unpredictable.

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 $\bar{3}$
1 2 3	9V9	1 2
1 2 3	9V9	2 3
+1.23	S9V99	1 2 $\overset{+}{3}$
+1.23	S9V9	1 $\overset{+}{2}$
123	9(5)	0 0 1 2 3
ZEROS	S99999	0 0 0 0 $\overset{+}{0}$
1 2 3 4 5	$**9.99	$ 1 2 3 . 4 5
1 2 3 4 5	999.9	1 2 3 . 4
0 0 0 1 2	$**9.99	$ * * 0 . 1 2

Any necessary conversion of data from one form of internal representation to another takes place during the move along with any specified editing in the receiving item.

A move that is not an elementary move is treated exactly as if it were an elementary alphanumeric to alphanumeric move. An index data item cannot appear as an operand in a MOVE statement.

If the CORRESPONDING option is used, selected items within identifier-1 are moved to selected items within identifier-2, according to the rules for the CORRESPONDING option. Results are the same as if the user had referred to each pair of corresponding identifiers in separate MOVE statements. Only one identifier may appear to the right of the word TO. Both identifier-1 and identifier-2 must be group items.

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

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.

4.7.14 MULTIPLY

The MULTIPLY statement causes numeric data items to be multiplied and sets the value of a data item or items equal to the results.

Format 1:

$\underline{\text{MULTIPLY}} \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \; \underline{\text{BY}} \text{ identifier-m } [\underline{\text{ROUNDED}}] \;\; , [\text{identifier-n}$

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

Format 2:

$\underline{\text{MULTIPLY}} \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \; \underline{\text{BY}} \; \left\{ \begin{array}{l} \text{identifier-2} \\ \text{literal-2} \end{array} \right\}$

$\underline{\text{GIVING}} \text{ identifier-m } [\underline{\text{ROUNDED}}] \;\; , [\text{identifier-n } [\underline{\text{ROUNDED}}]] \ldots$

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

In format 1, each identifier must refer to a numeric elementary item. In format 2, an identifier to the right of the word GIVING may refer to a data item that contains editing symbols. Each literal must be a numeric literal. The maximum size of each operand is 18 decimal digits. The initial value of identifier-1 or literal-1 is multiplied by the initial values of identifier-m, identifier-n,... The resulting products replace the values of identifier-m, identifier-n, ... respectively.

In format 2, the initial value of identifier-1 or literal-1 is multiplied by identifier-2 or literal-2 and the result is stored in identifier-m, identifier-n,... respectively.

Examples:

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

2. MULTIPLY ALPHA BY BETA GIVING CHI ROUNDED DELTA ROUNDED ON SIZE ERROR PERFORM ERROR-ROUTINE.

4.7.15 NOTE

The NOTE sentence allows the user to write commentary which will be produced on the listing, but not compiled.

NOTE character-string.

Any combination of the characters from the allowable character set may be included in the character string.

If a NOTE sentence is the first in a paragraph, the entire paragraph is considered to be part of the character string. Format rules for paragraph structure must be observed.

If a NOTE sentence is not the first sentence of a paragraph, the commentary ends with the first instance of a period followed by a space.

Examples:

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

2.  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.
```

4.7.16 OPEN

The OPEN statement initiates the processing of both input and output files. It performs checking and/or writing of labels and other input-output operations.

```
       / INPUT file-name-1 [ { REVERSED        } ] [ , file-name-2      \
       |                   [ { WITH NO REWIND  } ] [                    |
       |                                                                |
       |      [ { REVERSED        } ] ] ...                             |
       |      [ { WITH NO REWIND  } ] ]                                 |
       |                                                                |
OPEN  <  OUTPUT file-name-1 [WITH NO REWIND]                             > ...
       |                                                                |
       |      [, file-name-2 [WITH NO REWIND]] ...                      |
       |                                                                |
       | I-O file-name-1 [, file-name-2] ...                            |
       |                                                                |
       \ INPUT-OUTPUT file-name-1 [, file-name-2] ...                   /
```

At least one of the options INPUT, OUTPUT, I-O must be specified; however, there should be no more than one instance of each option per statement. These options may appear in any order. The I-O and INPUT-OUTPUT options pertain only to mass storage files.

All files except sort files must be opened with an OPEN statement prior to execution of the first SEEK, READ, WRITE or CLOSE statement for that file. Sort files must not be named in an OPEN statement. OPEN does not obtain or release the first data record; a READ must be executed to obtain, or a WRITE to release the first data record. A second OPEN statement for a file cannot be executed prior to the execution of a CLOSE statement for that file.

The user's beginning label subroutine is executed for checking or writing the first label if one is specified by a USE statement.

The REVERSED and NO REWIND options can be used only with sequential single-reel files; they do not apply to mass storage processing. If the external device permits rewinding, and if REVERSED or NO REWIND are not specified, execution of the OPEN statement positions the file at its beginning.

If either REVERSED or NO REWIND is specified, the file is not repositioned. REVERSED assumes the file is positioned at its end; NO REWIND assumes the file is positioned at its beginning.

When REVERSED is specified, subsequent READ statements make the data records of the file available in reverse order; that is, starting with the last record. Caution: The format of files to be read in reverse must be either unblocked logical records or blocked fixed-length logical records.

If OPTIONAL is specified for an input file in the File-Control paragraph of the Environment Division, the input-output control system checks for the presence or absence of this file. If the file is not present, the first READ statement for this file causes the imperative statement in the AT END phrase to be executed.

The I-O option permits the opening of a mass storage file for both input and output operations. Since this option implies the existence of the file, it cannot be used if the mass storage file is being created.

When the access mode is sequential, the OPEN statement supplies the initial address of the first record to be accessed in mass storage files.

When an attempt is made to open a mass storage file which the MASTER operating system indicates does not exist, the input-output system will attempt to allocate the file internally only if a FILE-LIMITS clause was specified for the file.

The following label information must be included in the FD entry for a file which is to be internally allocated:

$$\left\{\begin{array}{l}\underline{\text{ID}}\\ \underline{\text{IDENTIFICATION}}\end{array}\right\} \text{ IS } \left\{\begin{array}{l}\text{data-name-1}\\ \text{literal-1}\end{array}\right\}$$

$$\left\{\begin{array}{l}\underline{\text{OWNER}}\\ \underline{\text{OWNER-ID}}\end{array}\right\} \text{ IS } \left\{\begin{array}{l}\text{data-name-2}\\ \text{literal-2}\end{array}\right\}$$

$$\underline{\text{EDITION-NUMBER}} \text{ IS } \left\{\begin{array}{l}\text{data-name-3}\\ \text{literal-3}\end{array}\right\}$$

$$\underline{\text{ACCESS-PRIVACY}} \text{ IS } \left\{\begin{array}{l}\text{data-name-4}\\ \text{literal-4}\end{array}\right\}$$

$$\underline{\text{MODIFICATION-PRIVACY}} \text{ IS } \left\{\begin{array}{l}\text{data-name-5}\\ \text{literal-5}\end{array}\right\}$$

The number of blocks to be allocated initially must be given as integer-1 of the FILE-LIMITS clause.

The size of each mass storage block is determined from the BLOCK CONTAINS clause and/or RECORD CONTAINS clause. If the clause BLOCK CONTAINS integer-1 TO integer-2 CHARACTERS is given, a block size of integer-2 characters is allocated.

If the clause BLOCK CONTAINS integer-1 TO integer-2 RECORDS and RECORD CONTAINS integer-3 TO integer-4 CHARACTERS are given, a block size equal to integer-2 times integer-4 characters is allocated.

In addition to the basic block size, eight characters are added to allow for a header used internally by the input-output control system. This header is never accessible to the user program.

Files assigned to TTY or CRT are assumed by MASTER to be I-O files when the device is opened.

Examples:

1. OPEN INPUT CARD-FILE.
2. OPEN OUTPUT PRINT-FILEA WITH NO REWIND PRINT-FILEB.
3. OPEN INPUT-OUTPUT MASTER-FILE.

4.7.17
PERFORM

The PERFORM statement is used to depart from the normal execution sequence to execute one or more procedures either a specified number of times or until a specified condition is satisfied and to return control to the normal sequence.

Format 1 (simple):

$$\underline{\text{PERFORM}}\ \text{procedure-name-1}\ [\underline{\text{THRU}}\ \text{procedure-name-2}]$$

Format 2 (TIMES):

$$\underline{\text{PERFORM}}\ \text{procedure-name-1}\ [\underline{\text{THRU}}\ \text{procedure-name-2}]$$

$$\left\{\begin{array}{l}\text{identifier-1}\\ \text{integer-1}\end{array}\right\}\ \underline{\text{TIMES}}$$

Format 3 (UNTIL):

$$\underline{\text{PERFORM}}\ \text{procedure-name-1}\ [\underline{\text{THRU}}\ \text{procedure-name-2}]$$

$$\underline{\text{UNTIL}}\ \text{condition-1}$$

Format 4 (VARYING):

$$\underline{\text{PERFORM}}\ \text{procedure-name-1}\ [\underline{\text{THRU}}\ \text{procedure-name-2}]$$

$$\underline{\text{VARYING}}\ \left\{\begin{array}{l}\text{index-name-1}\\ \text{identifier-1}\end{array}\right\}\ \underline{\text{FROM}}\ \left\{\begin{array}{l}\text{index-name-2}\\ \text{literal-2}\\ \text{identifier-2}\end{array}\right\}$$

$$\underline{\text{BY}}\ \left\{\begin{array}{l}\text{literal-3}\\ \text{identifier-3}\end{array}\right\}\ \underline{\text{UNTIL}}\ \text{condition-1}$$

$$\left[\underline{\text{AFTER}}\ \left\{\begin{array}{l}\text{index-name-4}\\ \text{identifier-4}\end{array}\right\}\ \underline{\text{FROM}}\ \left\{\begin{array}{l}\text{index-name-5}\\ \text{literal-5}\\ \text{identifier-5}\end{array}\right\}\right.$$

$$\underline{\text{BY}}\ \left\{\begin{array}{l}\text{literal-6}\\ \text{identifier-6}\end{array}\right\}\ \underline{\text{UNTIL}}\ \text{condition-2}$$

$$\left[\underline{\text{AFTER}}\ \left\{\begin{array}{l}\text{index-name-7}\\ \text{identifier-7}\end{array}\right\}\ \underline{\text{FROM}}\ \left\{\begin{array}{l}\text{literal-8}\\ \text{identifier-8}\\ \text{index-name-8}\end{array}\right\}\right.$$

$$\underline{\text{BY}}\ \left\{\begin{array}{l}\text{literal-9}\\ \text{identifier-9}\end{array}\right\}\ \underline{\text{UNTIL}}\ \text{condition-3}]]$$

Each procedure name is the name of a section or paragraph in the Procedure Division.

Each identifier represents a numeric elementary item described in the Data Division. In format 2, the identifier represents a numeric item with no positions to the right of the assumed decimal point.

A literal must be numeric.

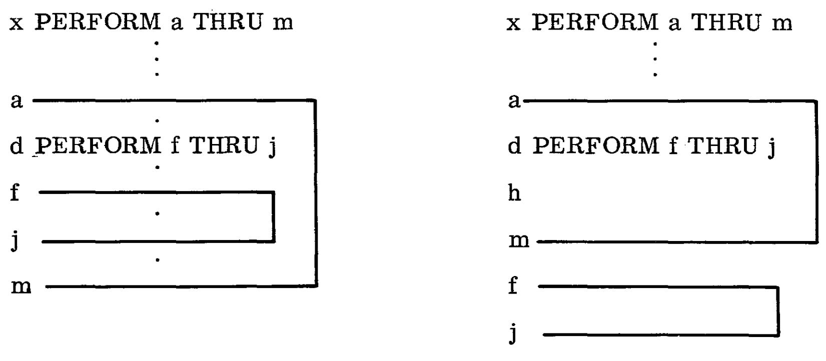
When the PERFORM statement is executed, control is transferred to the first statement of procedure-name-1. An automatic return is made to the statement following the PERFORM statement. If the THRU option is not specified, the return follows execution of the last statement of the paragraph or section referred to by procedure-name-1. If the THRU option is specified, the return follows execution of the last statement of the paragraph or section referred to by procedure-name-2.

The only relationship necessary between procedure-name-1 and procedure-name-2 is that a consecutive sequence of operations is to be executed beginning at procedure-name-1 and ending with the last statement in procedure-name-2. GO TO and PERFORM statements may occur between these limits. If there are two or more direct paths to the return point, then procedure-name-2 may name a paragraph containing an EXIT statement, to which all these paths must lead.

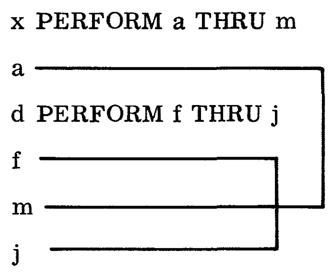
If control passes to these procedures by means other than a PERFORM statement, it passes through the last statement of the procedure to the following statement as if no PERFORM statement mentioned these procedures.

A procedure referenced by a PERFORM statement can include within it another PERFORM statement only if the procedure associated with the second PERFORM is entirely within or entirely without the procedure referenced by the first PERFORM.

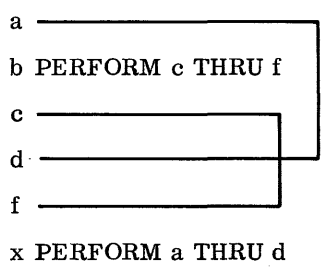
<u>Correct Specifications</u>



Incorrect Specifications



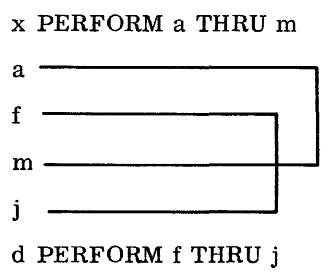
```
x PERFORM a THRU m
a
d PERFORM f THRU j
f
m
j
```



```
a
b PERFORM c THRU f
c
d
f
x PERFORM a THRU d
```

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

Correct Specification



```
x PERFORM a THRU m
a
f
m
j
d PERFORM f THRU j
```

Format 1:

A procedure referenced in the basic PERFORM statement is executed once and then control passes to statement following the PERFORM. For example:

```
PERFORM SUMMARY.
```

Format 2:

When the TIMES option is used, the procedures are performed the number of times specified by the value of integer-1 or the initial value of identifier-1. At the time of execution, this value must not be negative. If the value is zero, control passes immediately to the statement following the PERFORM. Otherwise, the procedures are executed the specified number of times and then control passes to the statement following the PERFORM. Once execution is begun, any change to identifier-1 has no effect on the number of times the procedures are executed.

Examples:

1. PERFORM CALCULATION 4 TIMES.
2. PERFORM ISSUE NO-COPIES TIMES.

Format 3:

If the UNTIL option is specified, the procedures are performed until the specified condition is true. Control returns to the statement following the PERFORM statement. If the condition is true when PERFORM is first encountered, the procedure is not executed.

Example:

```
PERFORM REORDER UNTIL ON-ORDER + SUPPLY = AV-USE * 2 or
SUPPLY GREATER THAN AV-USE * 2.
```

Format 4:

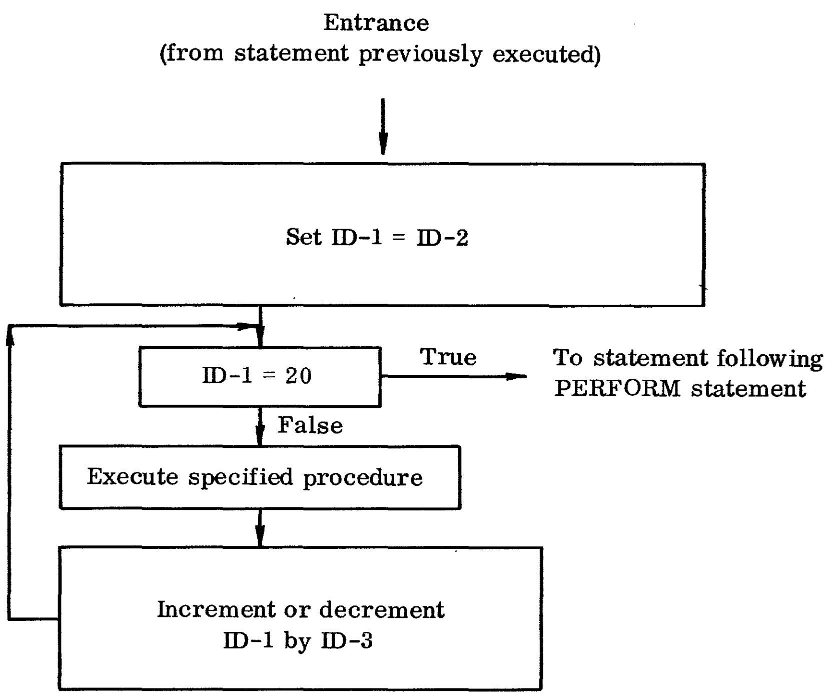
The VARYING option augments the value of one or more identifiers or index names during execution of the PERFORM statement. Every reference to an identifier as the object of the VARYING and FROM phrases also refers to index names. When index names are used, the FROM and BY clauses have the same effect as in a SET statement.

When one identifier is varied, identifier-1 is set equal to the value of identifier-2 or literal-2 at the start of execution of the PERFORM. The condition is evaluated, and if false, the sequence of procedures (procedure-name-1 through procedure-name-2) is executed once. Then the value of identifier-1 is augmented by the increment or decrement specified by identifier-3, and condition-1 is evaluated again. The cycle continues until the condition is true; then control passes to the statement following the PERFORM. If the condition is true at the start of execution, control passes directly to the statement following PERFORM.

Examples of a PERFORM statement with one varying identifier:

1. PERFORM ABLE THRU BAKER VARYING ID-1 FROM ID-2 BY ID-3 UNTIL ID-1 = 20.
2. PERFORM RATE-CALC VARYING QUAN FROM 50 BY 5 UNTIL QUAN GREATER 200.

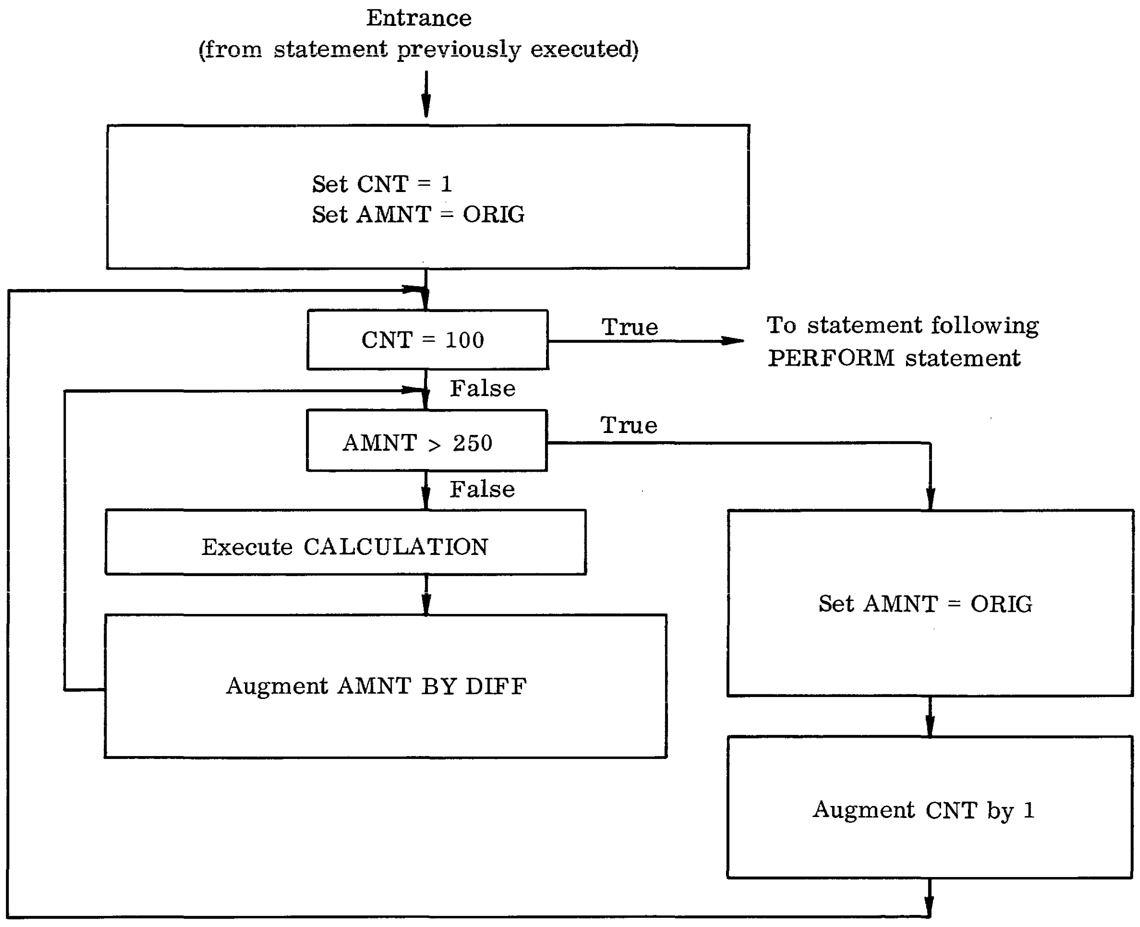
The first example is illustrated in the following flow chart:


Entrance
(from statement previously executed)
Set ID-1 = ID-2
ID-1 = 20
True
To statement following PERFORM statement
False
Execute specified procedure
Increment or decrement ID-1 by ID-3

When the optional clause beginning with AFTER is included, two identifiers are varied. Identifier-1 and identifier-4 are set to the values of identifier-2 and identifier-5, respectively. (During execution, these initial values must be positive). At the start of execution, condition-1 is evaluated. If true, control is transferred to the statement following the PERFORM. If false, condition-2 is evaluated. If condition-2 is false, procedure-name-1 through procedure-name-2 is executed once, after which identifier-4 is augmented by identifier-6 and condition-2 is evaluated again. This cycle of execution and augmentation continues until condition-2 is true. When it is true, identifier-4 is set to its initial value (identifier-5), identifier-1 is augmented by identifier-3, and condition-1 is re-evaluated. The PERFORM statement is completed if condition-1 is true; if not, the cycles continue until condition-1 is true. The identifiers following BY (identifier-3 and identifier-6) must not be zero.

Example of a PERFORM varying two identifiers:

```
PERFORM CALCULATION VARYING CNT FROM 1 BY 1 UNTIL
CNT = 100 AFTER AMNT FROM ORIG BY DIFF UNTIL AMNT
GREATER LIMT.
```

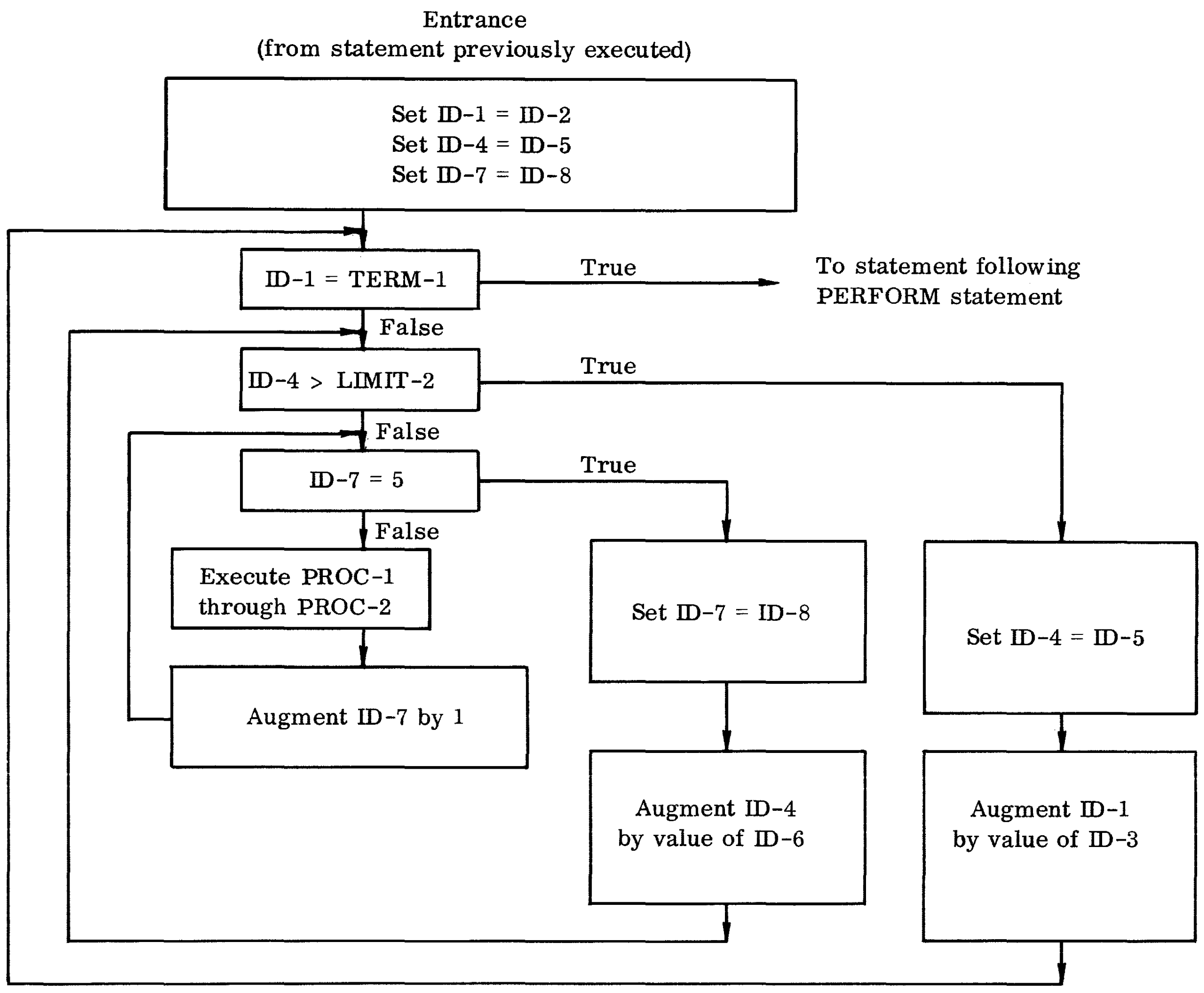


At termination of the PERFORM statement, AMNT contains its initial value. CNT has a value that exceeds the last used setting by an increment or decrement, unless condition-1 was true when the PERFORM statement was entered, in which case CNT and AMNT contain their initial values.

For three identifiers, the mechanism is the same as for two identifiers except that identifier-7 goes through a complete cycle each time that identifier-4 is augmented by identifier-6 or literal-6, which in turn goes through a complete cycle each time identifier-1 is varied.

Example of PERFORM varying three identifiers:

```
PERFORM PROC-1 THRU PROC-2 VARYING ID-1 FROM ID-2 BY ID-3
UNTIL ID-1 EQUAL TO TERM-1 AFTER ID-4 FROM ID-5 BY ID-6
UNTIL ID-4 GREATER LIMIT-2 AFTER ID-7 FROM ID-8 BY 1 UNTIL
ID-7 = 5.
```


Entrance
(from statement previously executed)
Set ID-1 = ID-2
Set ID-4 = ID-5
Set ID-7 = ID-8
ID-1 = TERM-1
True
To statement following PERFORM statement
False
ID-4 > LIMIT-2
True
False
ID-7 = 5
True
False
Execute PROC-1 through PROC-2
Set ID-7 = ID-8
Set ID-4 = ID-5
Augment ID-7 by 1
Augment ID-4 by value of ID-6
Augment ID-1 by value of ID-3

Upon completion of the PERFORM, ID-4 and ID-7 contain their initial values, ID-1 has a value exceeding its last used setting by one increment or decrement. If ID-1 equals TERM-1 (condition-1 is true) when the PERFORM statement is entered, ID-1, ID-4, and ID-7 all contain their initial values.

4.7.18 READ

The READ statement makes the next logical record available from an input file and allows performance of a specified imperative statement when end of file is detected.

Format 1: (for sequential files from any device)

```
READ file-name RECORD [INTO identifier-1]

        ; AT END imperative-statement
```

Format 2: (for random access mass storage files only)

```
READ file-name RECORD [INTO identifier-1]

        ; INVALID KEY imperative-statement
```

An OPEN statement must be executed for a file prior to execution of the first READ statement for that file.

If INTO is specified, the data is read into both the record area and the area specified by identifier-1.

Format 1:

If the logical end of the file is reached during execution of a READ, and an attempt is made to read that file, the AT END imperative statement is executed. Files assigned to TTY and CRT have no end-of-file condition, so the AT END phrase is never executed. Following execution of the AT END imperative statement, a READ statement for that file must not be given unless a prior CLOSE and OPEN for the file have been executed.

When a file consists of more than one type of logical record, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. Only the information present in the current record is accessible.

If a file described with the OPTIONAL clause is not present, the imperative statement in the AT END phrase is executed on the first READ and standard end-of-file procedures are not performed.

If the end of a tape reel is recognized during execution of a READ statement on a multi-reel/unit file, the following operations are carried out:

- Standard ending reel label procedure and user's ending reel label procedure, if specified, by the USE statement. The order of execution is specified by the USE statement.

- Tape swap.

- Standard beginning reel label procedure and user's beginning reel label procedure, if specified. The order of execution is specified by the USE statement.

- First data record on the new reel is made available.

Format 2:

The READ statement implicitly performs the functions of a SEEK statement for a specific mass storage file, unless a SEEK statement for the file is executed prior to the READ. Records in the file are read in accordance with the contents of the ACTUAL KEY data item. The user is responsible for setting the contents of this data item prior to execution of the READ statement (ACTUAL KEY, chapter 2). If an attempt is made to read a mass storage record and contents of the associated ACTUAL KEY data item are outside the allocated file area, the INVALID KEY phrase is executed. The allocated area of a file is the number of blocks assigned to it by the ALLOCATE request of the MASTER *DEF function (section 7.3.3).

Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged in that a record is available prior to the execution of any statement following the READ statement.

Examples:

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

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

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

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

4.7.19 RELEASE

The RELEASE statement transfers records to the initial phase of a SORT operation. It is equivalent to a WRITE statement applied to a sort file.

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

A RELEASE statement is used only within the range of an input procedure associated with a SORT statement for a sort file the description of which includes a DATA RECORDS clause containing record name.

RELEASE causes the record named by record name to be released to the initial phase of a sort. RELEASE must be part of an input procedure in the SORT statement. When control passes from the input procedure, the sort file consists of all records placed in it by execution of RELEASE statements. Following execution of the RELEASE statement, the contents of the record area named by record name are no longer available.

If the FROM option is included, the contents of the identifier data area are moved to record name before being released to the sort file. The move is made in accordance with the rules governing a simple MOVE without the CORRESPONDING option. The information in the data area associated with identifier remains available even though the information in the record name area is no longer available.

4.7.20 RETURN

The RETURN statement obtains sorted records from the final phase of a sort operation. It is equivalent to a READ statement applied to a sorted file.

```
RETURN file-name RECORD [INTO identifier]

    ; AT END imperative-statement.
```

A RETURN statement is used only within the range of an output procedure associated with a SORT statement for the specified file name. File-name is a sort file with a Sort File Description entry in the Data Division.

The INTO option is used only when the input file contains just one type of record. The identifier is the name of a working storage area or output record area. When INTO is specified, the data is available in both the input record area and the data area associated with identifier.

When a file consists of more than one type of record, records share the storage area automatically. This is equivalent to implicit redefinition of the area, and only the information that is present in the current record is accessible.

Execution of RETURN causes the next record, in the order specified by the keys listed in the SORT statement, to be made available for processing in the record area associated with the sort file. Moving is performed according to the rules specified for the MOVE statement without the CORRESPONDING option. After execution of the imperative statement in the AT END phrase, no more RETURN statements can be executed within the current output procedure.

4.7.21 SEARCH

The SEARCH statement is used to search a table for an element that satisfies the specified condition and to adjust the associated index name or index data item to point to that table element. (See Table Handling, section 4.5, also OCCURS, section 3.5.6, and SET, section 4.7.2.3.)

Format 1 (linear search):

```
SEARCH identifier-1 [VARYING {index-name-1}]
                             {identifier-2}

[; AT END imperative-statement-1]

; WHEN condition-1 {imperative-statement-2}
                   {NEXT SENTENCE         }

[; WHEN condition-2 {imperative-statement-3}] ...
[                   {NEXT SENTENCE         }]
```

Format 2 (bisecting search):

```
SEARCH ALL identifier-1 [; AT END imperative-statement-1]

; WHEN condition-1 {imperative-statement-2}
                   {NEXT SENTENCE         }
```

BINARY [illegible]

Identifier-1 cannot be subscripted or indexed. Its description must contain an OCCURS clause with the INDEXED BY option. In format 2, identifier-1 must also contain the KEY IS option in its OCCURS clause. When format 1 is used, condition-1, condition-2, etc, are any conditions defined in section 4.3. In format 2, condition-1 is limited to a relation condition using the relational operator EQUAL TO (=), a condition-name condition, or a compound condition formed by joining the first two conditions with an AND.

If index-name-1 is specified in the VARYING option, it must appear in the INDEXED BY option of an OCCURS clause describing either the table associated with identifier-1 or another table. If index-name-1 is in the INDEXED BY option of another table, the first (or only) index-name given in the INDEXED BY option for identifier-1 is used and index-name-1 is simultaneously incremented by the element character offset of the other table.

If identifier-2 is specified in the VARYING option, it must be either an index data item described with a USAGE IS INDEX clause or an elementary numeric item with no positions to the right of the assumed decimal point.

Format 1:

A simple SEARCH statement defines a linear search (a serial type of search operation). The search starts by checking the character address bias of the first (or only) index name listed in the INDEXED BY option for this table. The character address bias is the value which, when added to the base address of a table, gives the character address of a particular element in the table. If this value is greater initially than the highest permissible location value for identifier-1, the search terminates.

If the AT END clause is specified, the imperative statement associated with AT END is executed; otherwise, control passes to the next sentence. If the character address bias is not greater than the highest permissible location value for identifier-1, the conditions specified in the SEARCH statement are evaluated in the order they are written.

If no condition is satisfied, the index name is incremented by the element character offset to reference the next table element. The element character offset is the number of character positions in a table element. This process is repeated until one of the conditions is satisfied or the character address bias of the index name exceeds the table limit by one or more entries.

If a condition is satisfied, the imperative statement associated with the condition is executed. The index name remains set at the character address bias of the entry which satisfied the condition.

If the table limit is exceeded before a condition is satisfied, the AT END imperative statement is executed; otherwise, control passes to the next sentence.

If the SEARCH statement references an identifier-1 in a group or hierarchy of groups described with an OCCURS clause, each group must have an associated index name. Index name settings are used throughout the SEARCH to refer to identifier-1 or items within it. Only the index name associated with identifier-1 is incremented by the SEARCH; the items identifier-2 or index-name-1 are also incremented if they are specified in the VARYING option.

The VARYING option permits the location of entries in an associated table simultaneously with location of entries in the primary table (identified by identifier-1). If index-name-1 appears in the INDEXED BY clause describing identifier-1, that index name is used for the search, otherwise the first (or only) index name given in the INDEXED BY clause of identifier-1 is used. However, if index-name-1 appears in the INDEXED BY clause of another table, index-name-1 is incremented by the character address bias of the other table.

The starting point of a linear search may be established by using the SET statement to preset values of the index names associated with the table. SET need not be used with all SEARCH statements since the starting point of a linear search depends on the current value of the index name associated with the table or the index name specified in the VARYING option of the SEARCH statement. Thus, one SET...SEARCH combination can establish a position within the table and subsequent SEARCH statements will begin with the last setting of the table's index name. The SET statement does not apply to a bisecting search since the initial setting of the index name is ignored in that type of search.

Format 2:

The SEARCH ALL statement provides a bisecting nonserial type of search operation. The initial setting of the index name for identifier-1 is ignored and the index name setting is varied as the bisecting search takes place. The setting value will not exceed the location value of the last element in the table, nor be less than the location value of the first element in the table.

If condition-1 is satisfied, the index points to the table entry that satisfied the condition and the associated imperative statement is executed. If this imperative statement does not terminate with a GO TO, control passes to the next sentence.

[illegible]

If condition-1 is not satisfied for any setting of the index within the range permitted for the table, the imperative statement associated with the AT END option is executed. If no AT END is specified, control passes to the next sentence. When condition-1 cannot be satisfied, the final setting of the index is unpredictable.

Any or all data names in the KEY clause of identifier-1 may appear as subjects or objects of a test, or they may be conditional variables with which the tested condition name is associated. All preceding data names in the KEY clause hierarchy must also be tested within condition-1. No other tests may appear in condition-1.

Examples:

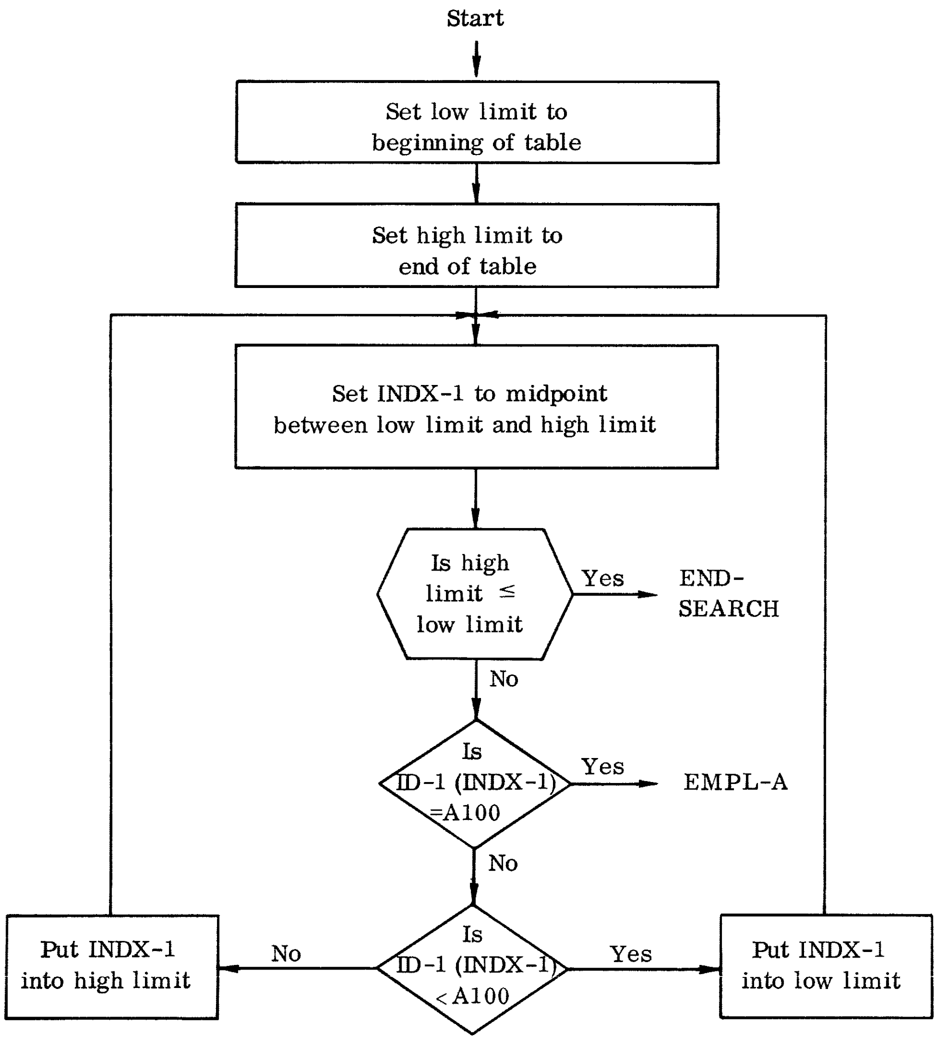
1. The following example shows a nonserial search of a one-dimensional table:

```
DATA DIVISION.
     .
     .
     .
01  TABLE-1.
    03  ID-1 PIC X(4) OCCURS 250 TIMES INDEXED BY INDX-1
        ASCENDING KEY IS ID-1.
     .
     .
     .
PROCEDURE DIVISION.
     .
     .
     .
```

```
SEARCH ALL ID-1 AT END GO TO END-SEARCH WHEN ID-1
    (INDX-1) = "A100" GO TO EMPL-A.
```

The bisecting search proceeds until an ID-1 is found with a value of "A100". If none is found, control passes to the procedure, END-SEARCH.

The search operation is illustrated below.



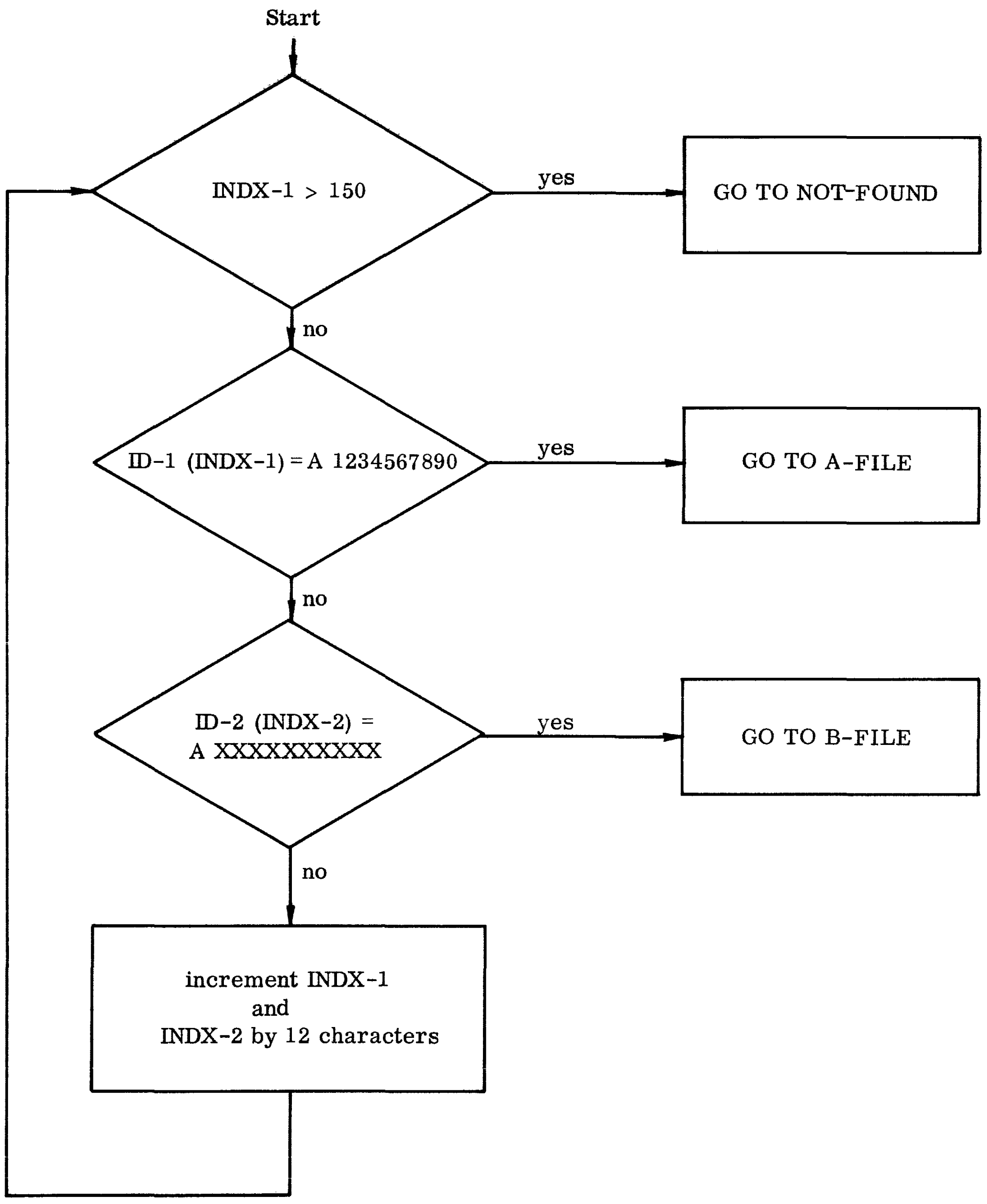
Start
Set low limit to beginning of table
Set high limit to end of table
Set INDX-1 to midpoint between low limit and high limit
Is high limit ≤ low limit
Yes
END-SEARCH
No
Is ID-1 (INDX-1) =A100
Yes
EMPL-A
No
Is ID-1 (INDX-1) <A100
No
Put INDX-1 into high limit
Yes
Put INDX-1 into low limit

2. The following example and flow chart illustrate a linear search with two WHEN conditional phrases:

```
DATA DIVISION.
          .
          .
          .
01  TABLE-1.
    03  ID-1 PIC X(10) OCCURS 150 TIMES INDEXED BY INDX-1.
01  TABLE-2.
    03  ID-2 PIC X(12) OCCURS 200 TIMES INDEXED BY INDX-2.
PROCEDURE DIVISION.
          .
          .
          .
    SEARCH ID-1 VARYING INDX-2 AT END GO TO NOT-FOUND
        WHEN ID-1 (INDX-1) = "A 1234567890" GO TO A-FILE
        WHEN ID-2 (INDX-2) = "A XXXXXXXXXX" GO TO B-FILE.
```

INDX-1 and INDX-2 are each incremented until one or the other of the two conditions is satisfied or until the character address bias of INDX-1 exceeds the table limit of TABLE-1. If neither condition is satisfied, control passes to the procedure NOT-FOUND.

INDX-1 and INDX-2 are each incremented until one or the other of the two conditions is satisfied or until the character address bias of INDX-1 exceeds the table limit of TABLE-1. If neither condition is satisfied, control passes to the procedure NOT-FOUND.



Start
INDX-1 > 150
yes
GO TO NOT-FOUND
no
ID-1 (INDX-1) = A 1234567890
yes
GO TO A-FILE
no
ID-2 (INDX-2) = A XXXXXXXXXX
yes
GO TO B-FILE
no
increment INDX-1 and INDX-2 by 12 characters

4.7.22 SEEK

The SEEK statement initiates access to a mass storage data record for subsequent reading or writing.

<u>SEEK</u> file-name RECORD

A SEEK statement pertains only to mass storage files in the random access mode and, when specified, is executed before each READ and WRITE statement.

SEEK finds the location of the record to be accessed from the contents of the ACTUAL KEY identifier. The user is responsible for setting the contents of this identifier prior to execution of the SEEK statement (see ACTUAL KEY, section 2.3.1). Validity of the contents of the ACTUAL KEY identifier for the particular mass storage file is determined at the time of execution. If the key is invalid, the imperative statement in the INVALID KEY clause of the next READ or WRITE statement for the associated file is executed.

Two SEEK statements for the same mass storage file may logically follow each other. Any validity check associated with the first SEEK statement is negated by execution of a second SEEK statement.

> CAUTION: The intent of the SEEK statement is to provide an overlap of actual processing while the read/write heads on a mass storage device are being positioned to the next data block that will be referenced by a subsequent READ or WRITE request. The advantage gained in the SEEK statement may be lost if other programs within the MASTER multiprogramming environment are accessing the same mass storage device. Therefore, the SEEK statement should be used only on random files that do not share mass storage devices with other files.

4.7.23 SET

The SET statement sets values for index names associated with elements in tables so that these table elements can be referenced by indexing.

Format 1:

$$\underline{\text{SET}} \left\{\begin{matrix}\text{index-name-1}\\ \text{identifier-1}\end{matrix}\right\} \left[\left\{\begin{matrix}\text{, index-name-2}\\ \text{, identifier-2}\end{matrix}\right\}\right] \ldots \underline{\text{TO}} \left\{\begin{matrix}\text{index-name-3}\\ \text{identifier-3}\\ \text{literal-1}\end{matrix}\right\}$$

Format 2:

$$\underline{\text{SET}} \text{ index-name-4 [, index-name-5]} \ldots \left\{\begin{matrix}\underline{\text{UP}}\ \underline{\text{BY}}\\ \underline{\text{DOWN}}\ \underline{\text{BY}}\end{matrix}\right\} \left\{\begin{matrix}\text{identifier-4}\\ \text{literal-2}\end{matrix}\right\}$$

The identifiers are index data items or numeric elementary items described without any positions to the right of the assumed decimal point. However, identifier-4 in format 2 may not be an index data item. A literal must be a positive integer. The index names must be defined in the INDEXED BY option of the OCCURS clause for a given table.

When SET is used to preset the contents of any index names in a table, all the index names which are to be set must be specified in the SET statement. No implicit setting is performed.

Format 1:

If index-name-1 is specified, it is set to the character address bias which corresponds to the occurrence number referred to by or contained in index-name-3, identifier-3 or literal-1. If identifier-3 is an index data item, or if index-name-3 is related to the same table as index-name-1, no conversion is made from occurrence number to character address bias.

If identifier-1 is specified, the particular action depends on whether it is an index data item or a data name containing a numeric elementary item. If it is an index data item, it is set equal to the contents of index-name-3 or identifier-3 which is also an index data item. Literal-1 cannot be used when identifier-1 is an index data item.

If identifier-1 is a data name, it is set to the occurrence number that corresponds to the character address bias in index-name-3. Neither identifier-3 nor literal-1 can be used in this case.

In the above discussion, all references to index-name-1 and identifier-1 apply to index-name-2 and identifier-2, respectively.

Format 2:

The contents of index-name-4 (and index-name-5, if specified) are incremented (UP BY) or decremented (DOWN BY) a value that corresponds to the number of occurrences represented by the value of literal-2 or identifier-4. Identifier-4 must be a data name identifying a numeric elementary item with no positions to the right of the assumed decimal point.

Example:

```
DATA DIVISION.
     77  B-1 PIC 99 USAGE IS INDEX.
     77  D-1 PIC 9  USAGE IS INDEX.
     01  ABLE.
          02  BAKER OCCURS 20 TIMES INDEXED BY INDX-B.
               03  CHARLIE PIC 9(3).
               03  DOG PIC X(3) OCCURS 5 TIMES INDEXED BY INDX-D.
.
.
.
PROCEDURE DIVISION.
M-1.     SET B-1 TO INDX-B.
M-2.     SET D-1 TO INDX-D.

         IF CHARLIE (B-1) LESS THAN 100,
              MOVE DOG (B-1, D-1) TO TABLE-D.
         IF INDX-B GREATER THAN 20; GO TO OUT.
         IF INDX-D LESS THAN 6; SET INDX-D UP BY 1; GO TO M-2;
              ELSE SET INDX-D DOWN BY 5, GO TO M-1.
OUT.
.
.
.
```

4.7.24 SORT

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

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

[      { DESCENDING }                              ]
[ ; ON {            }  KEY  {identifier-2} ...     ] ...
[      { 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                                       }
```

A program may contain more than one SORT statement in the Procedure Division. No SORT statement may appear in the Declaratives Section or in the input and output procedures associated with a SORT statement.

File-name-1 is a sort file described in an SD entry in the Data Division. 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-3 names an output procedure. File-name-2 and file-name-3 are described in FD entries in the Data Division. They are not sort files, and must not be described in an SD entry. The FILE-CONTROL paragraph of the Environment Division must specify only the SELECT and ASSIGN clauses for a sort file (file-name-1). FILE-CONTROL for file-name-2 and file-name-3 may include MULTIPLE and RESERVE as well as SELECT and ASSIGN.

ASCENDING specifies a sorting sequence from the lowest to the highest value of KEY; DESCENDING specifies a highest to lowest sequence. The order of values is in accordance with the ordered character set given in appendix D.

The record description of every record listed in the DATA RECORDS clause of the sort file must contain the KEY items identifier-1, identifier-2, etc. When KEY items appear in more than one record, data descriptions must be equivalent and starting positions must be the same character positions relative to the beginning of each record. They may not contain or be subordinate to entries that contain an OCCURS clause. The KEY identifiers are listed from left to right in the SORT statement in order of decreasing significance without regard to how they are divided into KEY clauses.

INPUT PROCEDURE consists of one or more sections written consecutively which must not be a part of any output procedure. An input procedure can include any procedures needed to select, create, or modify records, and at least one RELEASE statement to transfer a record to the sort file. Control may be passed to an input procedure only when a related SORT statement is being executed. If INPUT PROCEDURE is specified, control is passed to the input procedure before file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section; and when the last statement in the input procedure has been executed, the records that have been released to the sort file are sorted.

OUTPUT PROCEDURE consists of one or more sections written consecutively which must not be a part of any input procedure. An output procedure may consist of any procedures required to select, modify, or copy the records that are returned one at a time in sorted order from the sort file. At least one RETURN statement must be included in an output procedure to make the sorted records available for processing. Control must not be passed to an output procedure unless a related SORT statement is being executed. Control passes to the output procedure after file-name-1 has been sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section. When the last statement has been executed, this mechanism terminates the sort and transfers control to the statement following the SORT statement.

The procedural statements in an INPUT PROCEDURE or an OUTPUT PROCEDURE are subject to the following restrictions:

- The procedure must not contain any SORT statements.
- The procedure must not transfer control to points outside the procedure.
- The remainder of the Procedure Division must not contain any transfers of control to points inside an input or output procedure.

If USING is specified, the SORT statement automatically performs the functions of the OPEN, READ, USE, and CLOSE statements for file-name-2. All the records in file-name-2 are transferred automatically to file-name-1 for sorting. File-name-2 must not be open at the time of execution of the SORT statement.

If GIVING is specified, all the sorted records in file-name-1 are transferred automatically to file-name-3. File-name-3 must not be open when the SORT statement is executed. Execution of the SORT statement automatically opens file-name-3 before the records are transferred and closes it after the last record in the sort file is returned.

Examples:

1.
```
DATA DIVISION.
FILE SECTION.
FD  GEN-FILE
    BLOCK CONTAINS 10 RECORDS
    LABEL RECORD IS OMITTED
    DATA RECORD IS GEN-REC.
    01  GEN-REC.
        02  IDENT-A PIC 9(8).
        02  IDENT-B.
            03 ID-B PIC 99.
        02  IDENT-C PIC X(20).
SD SORT-FILE.
    DATA RECORD IS SORT-REC.
    01  SORT-REC.
        02  IDENT-1 PIC 9(8).
        02  IDENT-2.
            03  ID-2A PIC 99.
        02  IDENT-3 PIC X(20).
    .
    .
    .

PROCEDURE DIVISION.
A-SORT SECTION.
ST-1.   SORT SORT-FILE ON ASCENDING KEY IDENT-1
                        ON DESCENDING KEY ID-2A, IDENT-3
              INPUT PROCEDURE IS INP-1
              OUTPUT PROCEDURE IS OUT-1.
        GO TO REST-OF-PROGRAM.
INP-1 SECTION.
        OPEN INPUT GEN-FILE.
I-1.    READ GEN-FILE AT END GO TO I-2.
        MOVE GEN-REC TO SORT-REC.
        RELEASE SORT-REC.
        GO TO I-1.
I-2.    CLOSE GEN-FILE.
OUT-1 SECTION.
        OPEN OUTPUT GEN-FILE.
O-1.    RETURN SORT-FILE RECORD  AT END GO TO O-2.
        MOVE SORT-REC TO GEN-REC.
        WRITE GEN-REC.
        GO TO O-1.
O-2.    CLOSE GEN-FILE.
REST-OF-PROGRAM SECTION.
    .
    .
    .
```

2. Given two files, FILE-1 and FILE-2, with data records identical to GEN-REC in example 1, and a sort file identical to SORT-FILE in example 1, the following SORT statement can be specified:

```
    SORT SORT-FILE ON ASCENDING KEY IDENT-1, IDENT-3
                   ON DESCENDING KEY ID-2A
         USING FILE-1
         GIVING FILE-2.
```

4.7.25 STOP

The STOP statement halts the object program permanently or temporarily.

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

The literal may be numeric, non-numeric, or any figurative constant, except ALL.

If the RUN option is used, the program is terminated and control is returned to its caller.

If the literal option is used, the literal is displayed on the console. The program is temporarily suspended until the operator responds. A response of any combination of characters other than NO, causes the object program to continue with the execution of the next sequential statement. The response NO is reserved for the option of terminating program execution.

If a STOP statement with the RUN option appears in an imperative sentence, it must appear as the only or last statement in a sequence of imperative statements.

4.7.26
SUBTRACT

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from one or more items equal to the results.

Format 1:

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

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

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

All literals or identifiers preceding the word FROM are added together and this total is subtracted from identifier-m, identifer-n, ... The differences are stored in the respective identifiers.

Format 2:

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

$$\underline{\text{GIVING}} \text{ identifier-n } [\underline{\text{ROUNDED}}] \left[\text{, identifier-o } [\underline{\text{ROUNDED}}] \right] \ldots$$

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

All literals or identifiers preceding the word FROM are added together, subtracted from literal-m or identifier-m, and the result stored as the new value of identifier-n, identifier-o, ...

Format 3:

$$\underline{\text{SUBTRACT}} \left\{ \begin{matrix} \underline{\text{CORRESPONDING}} \\ \underline{\text{CORR}} \end{matrix} \right\} \text{ identifier-1 } \underline{\text{FROM}} \text{ identifier-2}$$

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

Data items in identifier-1 are subtracted from and stored into corresponding data items in identifier-2.

Each identifier must refer to a numeric elementary item except in format 2, where an identifier which appears to the right of the word GIVING may refer to a data item that contains editing symbols. Each literal must be a numeric literal.

The maximum size of each operand is 18 decimal digits; and the composite of operands, must not contain more than 18 digits. The composite of operands is the data item resulting from superimposing all operands aligned by decimal point, excluding the data items that follow the word GIVING.

The compiler insures that enough places are carried to avoid loss of significant digits during execution.

Examples:

```
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.
```

4.7.27 TRACE

The TRACE statement is a source language debugging tool which enables the user to display literals and the contents of data names on the job standard output file during execution of the object program.

$$\underline{\text{TRACE}}\ [\underline{\text{WHEN}}\ \text{integer-1}]\ \ [\underline{\text{EVERY}}\ \text{integer-2}]\ \ [\underline{\text{UNTIL}}\ \text{integer-3}]$$

$$\begin{Bmatrix}\text{literal-1}\\ \text{data-name-1}\end{Bmatrix}\left[,\begin{Bmatrix}\text{literal-2}\\ \text{data-name-2}\end{Bmatrix}\right]\ldots.$$

Any number of TRACE statements may be inserted in the Procedure Division of a COBOL source program.

Compilation of TRACE statements is controlled by the TRACE parameter on the UCBL control card for the program. When the T parameter is specified, all TRACE statements are compiled into the object program at the points of occurrence.

During program execution, as each compiled TRACE request is encountered, each specified literal value is displayed as given. The contents of data names are displayed in edited form according to the edit picture assigned to each data name. The data name followed by a space, an equal sign and another space are presented preceding the value. All literals and values are displayed in a linear fashion across the print line.

WHEN integer-1 indicates the first execution of the TRACE statement. Each time the program enters a TRACE request, a counter initially set to one is incremented by one and a comparison is made with the value given as integer-1. When the counter equals integer-1, the literals and values for the request are displayed for that cycle and all successive integer-2 cycles, until integer-3 has been reached. If WHEN integer-1 is omitted, the TRACE begins at the first cycle.

EVERY integer-2 indicates the timing between each display. If this option is omitted, a display is generated during each cycle of the TRACE request.

UNTIL integer-3 indicates the maximum number of TRACE cycles to be processed before the request becomes inoperative. If the option is omitted, the TRACE request is processed until program execution is ended.

A TRACE statement may be continued on succeeding source lines, it may begin on or after column 12.

Example:

```
     READ FILE-A AT END GO TO FINISH.
     TRACE WHEN 50 EVERY 10 UNTIL 5000 "FILE-A"
     REC-COUNT, REC-ID, "THE FOLLOWING" REC-SIZE.
```

The result of the TRACE statement is:

```
     FILE-A REC-COUNT = 1,500 REC-ID = INV-MAST THE
          FOLLOWING REC-SIZE = 150
```

4.7.28 USE

The USE statement introduces user-defined input-output label and error handling procedures.

Format 1:

```
                         ( file-name-1 [file-name-2] ... )
                         | INPUT                         |
USE AFTER STANDARD       < OUTPUT                        > .
ERROR PROCEDURE ON       | I-O                           |
                         ( INPUT-OUTPUT                  )
```

Format 2:

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

                       ( file-name-1 [file-name-2] ... )
                       | INPUT                         |
   LABEL PROCEDURE ON  < OUTPUT                        > .
                       | I-O                           |
                       ( INPUT-OUTPUT                  )
```

Format 3:

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

A USE statement immediately follows a section header in the Declaratives portion of the Procedure Division. It must be followed by a period and a space. The remainder of the section consists of one or more procedural paragraphs defining the procedures to be used.

The USE statement itself is never executed, rather it defines the conditions calling for execution of the USE procedures.

A USE procedure must not contain any reference to non-declarative procedures. Conversely, the non-declarative portion must not contain references to procedure names that appear in the declarative portion; except that a PERFORM statement may refer to a USE declarative or to the procedures associated with a USE declarative.

The only I-O statements allowed within a USE procedure are ACCEPT, DISPLAY, and CLOSE. A file name must not be referred to explicitly or implicitly in more than one USE statement. No file name referred to in a USE procedure can represent a sort file.

The USE procedures in format 1 are executed by the input-output control system after completion of the standard input-output error routine.

If a bad block is read, the error declarative is executed for each READ statement that causes retrieval of a logical record from the bad block. On entry to the declarative, the record requested will have been moved to the record area. Upon exiting the declarative, the input-output control system ignores the error, accepts the bad record, and resumes normal processing.

If a WRITE error occurs, the declarative is executed only once for the error.

In format 2 for an input file, the procedures are executed before and/or after execution of a beginning or ending input label check procedure. For an output file, the USE procedures are executed either before a beginning or ending output label is created, or after a beginning or ending output label is created but before it is written on tape.

The following rules govern format 2 USE procedures:

> If file-name-1, file-name-2, etc. is specified, the associated File Description entry for the file must not specify LABEL RECORDS ARE OMITTED.
>
> If the words BEGINNING or ENDING are not included, the designated procedures are executed for both beginning and ending labels.
>
> If neither REEL nor FILE is included, the designated procedures are executed for both REEL and FILE labels. The REEL option is not applicable to mass storage files.

The USE label procedures are not applicable to scratch files. The USE label procedure statement may be used for mass storage files even though mass storage file labels are maintained by MASTER and are not available to the user. Only the BEFORE or AFTER BEGINNING FILE or UNIT options of format 2 are valid. The UNIT and FILE options are synonymous. The input-output control system passes control to the BEFORE BEGINNING procedure just before the user's label identification data is moved to a MASTER OPEN request skeleton. The MASTER OPEN request makes the file available for processing, and then control passes to the AFTER BEGINNING label procedure.

USE BEFORE REPORTING (format 3) is used only in conjunction with the Report Writer. It is described in section 5.4.4 with the Report Writer statements.

4.7.29 WRITE

The WRITE statement releases a logical record from the output record area to the input-output control system which writes the record on an external device as part of an output file.

Format 1:

```
WRITE record-name [FROM identifier-1]

⎡⎧BEFORE⎫             ⎧identifier-2 LINES⎫⎤
⎢⎨      ⎬ ADVANCING  ⎨integer LINES     ⎬⎥
⎣⎩AFTER ⎭             ⎩mnemonic-name     ⎭⎦
```

Format 2:

```
WRITE record-name [FROM identifier-1]

     ; INVALID KEY imperative-statement
```

Format 1 is used to process non-mass storage files. The ADVANCING option allows the user to specify vertical position of the printer. Format 2 is used to process mass storage files. INVALID KEY specifies an imperative statement to be executed if an attempt is made to write outside the allocated file area.

An OPEN statement must be executed for a file before a WRITE statement for that file can be executed.

Record name is the name of a logical record in the File Section of the Data Division; it must not name a sort file. After the WRITE is executed, record name is no longer available.

The FROM option is used to move a record from the area in memory specified by identifier-1 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. Record name may not be the same as identifier-1.

Format 1:

When WRITE is specified for a multi-reel tape file, the following operations are performed after an end-of-reel condition is recognized:

Standard ending reel label procedure and user's ending reel label procedure (if specified by a USE statement) in the order specified by the USE statement.

Reel swap.

Standard beginning reel label procedure and user's beginning reel label procedure (if specified by a USE statement) in the order specified by the USE statement.

When records are written for a teletypewriter or 211 display unit, the user is responsible for determining the positioning of the data. The input-output control system interprets the first character of the file's record area before the write operation is initiated. If the first character is the internal octal code 36, the following action will take place before the record is displayed on the device:

Device	Action
TTY	Carriage return to the lefthand margin of the page. The first character of the record is not printed.
CRT	Clear the screen and position the entry marker to the upper lefthand corner of the screen. The first character of the record is converted to a blank character and is displayed as such.

When the first character of the record is other than internal code 36, the record is displayed immediately following the last character written by a previous WRITE request. In this case, the first character is always displayed as presented by the WRITE request.

The ADVANCING option allows control of the vertical positioning of each record on the printed page. The ADVANCING option overrides automatic advancing.

Identifier-2 must be the name of a numeric elementary item described without any positions to the right of the assumed decimal point. The mnemonic name is defined in the Special-Names paragraph of the Environment Division; it is identified with a printer carriage control character.

- If identifier-2 is specified, the printer page is advanced the number of lines contained in identifier-2.
- If integer is specified, the printer page is advanced the number of lines equal to the value of integer.
- If mnemonic-name is specified, the printer page is advanced according to printer carriage control rules below:

Control Character	Action Before Print	Action After Print
A	Space 1	Page Eject
B	Space 1	Skip to Last Line
C	Space 1	Skip to Level 6
D	Space 1	Skip to Level 5
E	Space 1	Skip to Level 4
F	Space 1	Skip to Level 3
G	Space 1	Skip to Level 2
H	Space 1	Skip to Level 1
1	Page Eject	No Space
2	Skip to Last Line	No Space
3	Skip to Level 6	No Space
4	Skip to Level 5	No Space
5	Skip to Level 4	No Space
6	Skip to Level 3	No Space
7	Skip to Level 2	No Space
8	Skip to Level 1	No Space
0 (zero)	Space 2	No Space
+	No Space	No Space
-	Space 3	No Space
(blank)	Space 1	No Space
Q	Clear Auto Page Eject	No Print
R	Select Auto Page Eject	No Print

When a file is assigned to PRINTER or SYSTEM-OUTPUT the first character (carriage control character) of the record is suppressed when the record is printed.

Format 2:

When WRITE is specified for mass storage files in the sequential access mode, the imperative statement in the INVALID KEY clause is executed when the end of the last allocated segment of the file is reached; and an attempt is made to execute a WRITE statement for that file. If the THRU integer-4 option of the FILE-LIMITS clause is specified, the file will be expanded by ten percent of the originally allocated area and the INVALID KEY imperative statement is not executed. However, if an attempt to expand the file results in the maximum limit (integer-4) being reached or all scheduled mass storage being used, the INVALID KEY imperative statement is executed.

When WRITE is specified for a mass storage file in random access mode, records are written on the file in accordance with the contents of the ACTUAL KEY data item. The user is responsible for setting the contents of this data item prior to execution of the WRITE statement (ACTUAL KEY, chapter 2). The imperative statement in the INVALID KEY phrase is executed when the contents of the ACTUAL KEY data item are out of range.

When an INVALID KEY condition exists, no writing takes place and the information in the record area remains available.

The WRITE statement performs the function of the SEEK statement unless a SEEK statement for the specified record is executed prior to the WRITE.

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 LINE-NO LINES.

WRITE ERROR-REC INVALID KEY ADD 1 TO ERR-COUNT GO TO
ERROR-PROC.
```

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. Once defined, the Report Writer statements in the Procedure Division place the report in the specified format on a user specified device. 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.

5.1 GENERAL DESCRIPTION

A report is a pictorial presentation of data. In preparing a report, the format id 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 is to be written.

Two types of entries are required for each report: the report description entry (RD level) which describes the physical aspects of the report format and the report group description entry (01 level) which describes the characteristics of the items included in the report and their relation to the report format.

The Report Description (RD level) specifies the overall format: characteristics of the page are outlined; limits are prescribed for the page and for footings, headings, and detail information. This entry also specifies data items that act as print control factors. Each report associated with an output file must be defined in an RD entry.

Each report must contain at least one report group. A report group is a set of one or more data items which is always presented as a single unit regardless of format. It may consist of one or more report lines. The report group always has an 01 level number; but it may contain lower level group and/or elementary items with level numbers from 2-49. Each report group at the 01 level must have a TYPE clause which specifies the type of the report group: heading, footing, or detail. A data name identifying the report group is optional. However, it must be specified for any report group referenced by GENERATE or USE statements in the Procedure Division. The placement of an item in relation to the report group and of the report group in relation to the entire report, the format description of all items, and any control factors associated with the report group are defined by this entry.

5.1.1 CONTROL GROUPS/ CONTROL BREAKS

Summary information can 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 entry 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. A control group is the set of control heading, control footing, and detail report groups associated with a given control data name. Within the control hierarchy, lower level heading and/or footing report groups are included in the higher level control group.

5.1.2 PAGE BREAK

A page break occurs whenever the LINE-COUNTER is changed, by a LINE NUMBER clause or a NEXT GROUP clause, to a line number that is not currently available to the user. A line number may not be available to the user under the following two conditions:

- Relative spacing exceeds the PAGE LIMIT clause specification
- Absolute spacing references a number that is equal to or less than the current line number

If the above page break conditions are detected at object time, PAGE HEADING and PAGE FOOTING report groups are produced if appropriate. The page counter associated with the fixed data name PAGE-COUNTER is incremented by one each time a page break occurs. It is incremented after the page footing is produced and before the page heading on the next page.

5.1.3 LINE-COUNTER

The fixed data name, LINE-COUNTER is generated automatically by the Report Writer for each report. LINE-COUNTER determines when a page heading and/or footing report group is to be presented and controls vertical spacing of information on the page. It is a numeric item, and its size is based on the number of lines per page specified in the PAGE LIMIT clause.

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 the LINE NUMBER and NEXT GROUP clauses in a report group description. The value of LINE-COUNTER may be incremented with relative spacing, or it may be replaced with absolute spacing. If relative spacing advances the line counter beyond the limits defined in the PAGE LIMIT clause, PAGE FOOTING is presented if appropriate and the line counter is reset to zero (top-of-form). No additional setting is made to the line counter based on the relative line specification. If absolute spacing causes the line counter to be less than or equal to the current line number, the line counter is set to the requested absolute line number following the generation of PAGE-FOOTING and PAGE-HEADING groups.

LINE-COUNTER may be referred to by Procedure Division statements; however, if it is changed by a Procedure Division statement, page format control may be unpredictable. If more than one line counter exists because more than one Report Description entry is specified, LINE-COUNTER must be qualified by a report name when it is referenced by a Procedure Division statement.

The value of LINE-COUNTER during execution represents:

- Last line number of previous report group
- Last line number skipped by a previous NEXT GROUP specification
- Zero, if at top of page

5.1.4 PAGE-COUNTER

PAGE-COUNTER is a fixed data name automatically generated by the Report Writer as a data item to number the pages within a report. One numeric page counter is supplied for each report. The size of the counter is specified by the PICTURE clause associated with the first elementary data item that uses PAGE-COUNTER as a source. If PAGE-COUNTER is given as a source in more than one item, each PICTURE clause must specify the identical number of numeric characters. The size must be sufficient to prevent overflow.

PAGE-COUNTER may be referenced from the Procedure Division. If more than one Report Description entry exists, PAGE-COUNTER must be qualified by the report name when referenced. It must not be qualified when used as a source in the Data Division.

The page counter is automatically preset to one by the Report Writer. If a starting value greater than one is desired, the contents of the page counter can be changed with a Procedure Division statement following the INITIATE statement.

The page counter is automatically incremented by one each time a page break occurs. It is incremented after page footing and before a page heading is generated.

5.2 DATA DIVISION ENTRY FORMATS

Format specifications used in the Report Writer are defined in this section. Each 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.

5.2.1 FILE DESCRIPTION ENTRY

The File Description furnishes information concerning the physical structure, identification, and record names of a given file.

```
FD file-name
┌                                                    ┌                ┐ ┐
│ ; BLOCK CONTAINS [integer-1 TO] integer-2          │ RECORDS        │ │
│                                                    │ CHARACTERS     │ │
└                                                    └                ┘ ┘
┌
│ ; RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS
└
      ┌                   { data-name-1   } ┐ ┐
      │ DEPENDING ON      { RECORD-MARK   } │ │
      └                                     ┘ ┘

  ; LABEL { RECORD IS    }  { STANDARD    }
          { RECORDS ARE  }  { OMITTED     }
                            { data-name-2 }
┌
│ ; VALUE OF { data-name-3  }  IS  { data-name-4 }
│            { label-field-1 }      { literal-1   }
└
      ┌ , { data-name-5   }  IS  { data-name-6 } ┐      ┐
      └   { label-field-2 }      { literal-2   } ┘ ...  ┘
┌                                                              ┐
│ ; DATA { RECORD IS   }  data-name-7  [,data-name-8] ...      │
│        { RECORDS ARE }                                       │
└                                                              ┘
  ; { REPORT IS   }  report-name-1 [, report-name-2] ... .
    { REPORTS ARE }
```

The level indicator FD identifies the file description entry; it must precede the file name. All semicolons are optional, but the entry must be terminated by a period.

Clauses which follow the file name are optional in many cases, and their order of appearance is immaterial. See section 3 for a discussion of the BLOCK, RECORD, LABEL, VALUE OF, and DATA clauses.

If the RECORD CONTAINS or DATA RECORD clauses are not specified, a record area 136 characters long is automatically assigned to the file.

The REPORT clause cross references the report description entries with associated file description entries:

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

The REPORT clause is required in the file description entry if the file is an output report or if it is to contain output report records. Each report name listed in the FD entry must be the subject of a report description (RD) entry in the Report Section. The presence of more than one report name indicates that the file contains more than one report. These reports need not have the same description and the order in which they are listed is not significant.

5.2.2 REPORT DESCRIPTION ENTRY

The Report Description defines the physical structure and identification of a named report.

Format 1:

```
RD report-name-1 [WITH CODE mnemonic-name] 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]

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

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

The level indicator RD which identifies the Report Description must precede the report name. The report name must appear in a REPORT clause of an FD entry. All semicolons are optional in the Report Description but the entry must be terminated by a period.

Clauses which follow the report name are optional in many cases. Except in format 1, the order of their appearance is immaterial.

The fixed data names, LINE-COUNTER, and PAGE-COUNTER are automatically generated by the Report Writer based on the presence of specific entries; they are not data clauses.

5.2.3 REPORT GROUP DESCRIPTION ENTRY

The report group description entry specifies the characteristics of a particular report group and of the individual items within a report group. All semicolons are optional in the report group description but the entry must be terminated by a period. A report group must have a data-name if it is referred to by a Procedure Division statement.

Format 1:

```
01 [data-name-1]

┌                  ( integer-1         ) ┐
│ ; LINE NUMBER IS { PLUS integer-2    } │
└                  ( NEXT PAGE         ) ┘

┌                 ( integer-3          ) ┐
│ ; NEXT GROUP IS { PLUS integer-4     } │
└                 ( NEXT PAGE          ) ┘

            ⎛ { REPORT HEADING  }                      ⎞
            ⎜ { RH              }                      ⎟
            ⎜ { PAGE HEADING    }                      ⎟
            ⎜ { PH              }                      ⎟
            ⎜ { CONTROL HEADING } { identifier-1 }     ⎟
            ⎜ { CH              } { FINAL        }     ⎟
; TYPE IS   ⎨ { DETAIL          }                      ⎬
            ⎜ { DE              }                      ⎟
            ⎜ { CONTROL FOOTING } { identifier-2 }     ⎟
            ⎜ { CF              } { FINAL        }     ⎟
            ⎜ { PAGE FOOTING    }                      ⎟
            ⎜ { PF              }                      ⎟
            ⎜ { REPORT FOOTING  }                      ⎟
            ⎝ { RF              }                      ⎠

[; USAGE IS DISPLAY].
```

Format 1 indicates a report group; the report group extends from this entry to the next report group (level 01) entry. The TYPE clause is required for every report group to indicate whether the group is a heading, detail, or footing group. If a control heading or control footing group is specified, TYPE also indicates the control item which will cause these groups to be generated.

Format 2:

$$\text{nn [data-name-1]}$$

$$[;\ \underline{\text{COLUMN}}\ \text{NUMBER IS integer-1}]$$

$$[;\ \underline{\text{GROUP}}\ \text{INDICATE}]$$

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

$$\left[;\ \underline{\text{LINE}}\ \text{NUMBER IS}\left\{\begin{array}{l}\text{integer-2}\\ \underline{\text{PLUS}}\ \text{integer-3}\\ \underline{\text{NEXT}}\ \underline{\text{PAGE}}\end{array}\right\}\right]$$

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

$$\left[;\ \underline{\text{RESET}}\ \text{ON}\left\{\begin{array}{l}\text{identifier-1}\\ \underline{\text{FINAL}}\end{array}\right\}\right]$$

$$[;\ \underline{\text{BLANK}}\ \text{WHEN}\ \underline{\text{ZERO}}]$$

$$\left\{\begin{array}{l};\ \underline{\text{SOURCE}}\ \text{IS identifier-2}\\ ;\ \underline{\text{SUM}}\ \text{identifier-3}\quad [,\ \text{identifier-4}]\ \ldots\ [\underline{\text{UPON}}\ \text{data-name-2}]\\ ;\ \underline{\text{VALUE}}\ \text{IS literal-1}\end{array}\right\}$$

$$[;\ \text{USAGE IS}\ \underline{\text{DISPLAY}}]\ .$$

Format 2 indicates an elementary item or group item within a report group. If a report group consists of only one elementary entry, format 2 may include TYPE and NEXT GROUP clauses to specify the report group and elementary item in the same entry. In this case, the level number must be 01. Otherwise, the level numbers in format 2 are in the range 02 to 49. Elementary item descriptions are written (left to right) in the order in which they will appear on the printed page. The position of a report group within a report is determined by the TYPE clause.

5.3 DATA DIVISION CLAUSES

See section 3 for a discussion of the JUSTIFIED, PICTURE, BLANK WHEN ZERO, and USAGE clauses. The remaining clauses in the report description entry and the report group description entry are described in the following pages; they appear in alphabetic order.

5.3.1 CODE

The CODE clause defines a unique character to be affixed to each report line produced in this report.

WITH CODE mnemonic-name-1

The CODE clause, when used in formal 1 of the RD entry, must follow immediately after the report name.

CODE mnemonic-name-1 indicates a unique character which is automatically prefixed to and identifies each report line produced. Multiple reports may then be produced simultaneously onto one output device for later individual report selection. The mnemonic name must appear in the Special-Names Paragraph of the Environment Division.

5.3.2 COLUMN NUMBER

This clause indicates the absolute column number on the printed page of the high-order (leftmost) character of the elementary item. Integer-1 must be positive.

<u>COLUMN</u> NUMBER IS integer-1

The COLUMN NUMBER clause can be given only at the elementary level within a report group. For a particular line, COLUMN NUMBER entries are presented in the order, from left to right, in which the items will appear on the page.

The COLUMN NUMBER clause must be specified if this elementary item is to be presented. If the column number is not indicated, the elementary item, though included in the description of the report group, is suppressed when the report group is produced. When COLUMN NUMBER is specified, the PICTURE clause and one of the clauses, SOURCE, SUM, or VALUE must be included in the item description.

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.
```

5.3.3 CONTROL

The CONTROL clause specifies the identifiers which provide control breaks for this report. Whenever the value contained in these identifiers changes, a control break occurs. The control hierarchy is defined by the order of the control breaks. This clause is included in the report description (RD) entry for an entire report.

$$\left\{\begin{array}{l}\underline{\text{CONTROL}}\text{ IS}\\ \underline{\text{CONTROLS}}\text{ ARE}\end{array}\right\}\left\{\begin{array}{ll}\underline{\text{FINAL}} & \\ \text{identifier-1} & [,\text{ identifier-2}]\ldots\\ \underline{\text{FINAL}},\text{ identifier-1} & [,\text{ identifier-2}]\ldots\end{array}\right\}$$

The CONTROL clause is required when control heading or control footing report groups are specified.

The identifiers are listed in order from major to minor; FINAL is the highest control, identifier-1 is the major control, identifier-2 is the intermediate control, etc. The last identifier specified is the minor control. The idenfi-fiers must be defined in the File, Common-Storage or Working-Storage Section of the Data Division.

The identifiers specified in the CONTROL clause are the only identifiers referred to by the RESET and TYPE clauses in a report group description entry for this report. This clause is optional, but must be included when the TYPE clause specifies control headings or footings or when RESET is specified. Control headings and footings are printed automatically whenever the value of an identifier specified in this clause changes. The final control heading is printed before any other control headings when the first GENERATE is executed, and final control footing is printed when the TERMINATE statement is executed.

Example:

```
RD  REPORT-A
    CONTROLS ARE FINAL, DEPT-NO, SECT-NO
        .
        .
        .
```

During the generation of REPORT-A, the control headings and footings associated with SECT-NO are produced when the contents of the data item called SECT-NO changes. The control headings and footings associated with DEPT-NO are produced in addition to those associated with SECT-NO when the contents of DEPT-NO changes. The final control heading, in addition to all lower level control headings, is produced when the first GENERATE statement is executed for this report, the final footing follows all other control footings when the TERMINATE statement is executed.

5.3.4 GROUP INDICATE

This clause specifies that its associated elementary item is to be presented only once on the first occurrence of the item following a control break or at the beginning of a new page.

GROUP INDICATE

GROUP INDICATE must be given only at the elementary item level within a TYPE DETAIL report group.

An elementary item is group indicated in the first DETAIL report group containing the item after a control break. It is also group indicated in the first DETAIL report group containing the item on a new page, even if a control break did not occur.

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 be printed only the first time the detail group appears on a page or when a control break occurs. The other items in the group will be printed each time the detail group is printed.

5.3.5 LINE NUMBER

This clause indicates the absolute or relative line number of the entry in reference to the page or the previous entry.

$$\underline{\text{LINE}}\ \text{NUMBER IS} \left\{ \begin{array}{l} \text{integer-1} \\ \underline{\text{PLUS}}\ \text{integer-2} \\ \underline{\text{NEXT}}\ \underline{\text{PAGE}} \end{array} \right\}$$

Integer-1 and integer-2 are positive integers. Integer-1 must be within the range specified by the PAGE LIMITS clause in the report description entry.

The LINE NUMBER clause must be given for each report line of a report group. For the first line of a report group it is given at the report group level; it may also be given before or at the first elementary item in the line. For report lines other than the first in a report group, it must be given before or at the first elementary item in the line.

If the LINE NUMBER clause is specified at the report group level, entries for the first report line within the report group are presented on the specified line number. If LINE NUMBER is specified for an entry at a subordinate level, all succeeding printable items are presented on that print line, until another LINE NUMBER clause is declared or the report group ends. Printable items are those elementary items which have descriptions containing the COLUMN NUMBER clause. A line number for the first subordinate level may not contradict the line number of its group level.

Integer-1 indicates an absolute line number. The line counter is set to this value in this and following entries within the report group until a different value for LINE NUMBER is specified.

PLUS integer-2 indicates a relative line number which increments the line counter in this and following entries within the report group until a different value for LINE NUMBER is specified.

Within a report group, absolute LINE NUMBER entries must be indicated in ascending order, and an absolute LINE NUMBER cannot be preceded by a relative LINE NUMBER.

The NEXT PAGE phrase indicates an automatic skip to the next page before presenting the first line of the current report group. Appropriate page footings and page headings will be produced as specified. NEXT PAGE may not be specified for page heading or page footing report groups.

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.

5.3.6 NEXT GROUP

The NEXT GROUP clause specifies the spacing between the last line of the report group and the next report group to be generated.

$$\underline{\text{NEXT}}\ \underline{\text{GROUP}}\ \text{IS} \left\{ \begin{array}{l} \text{integer-1} \\ \underline{\text{PLUS}}\ \text{integer-2} \\ \underline{\text{NEXT}}\ \underline{\text{PAGE}} \end{array} \right\}$$

The NEXT GROUP clause must appear only at the 01 level which defines the report group.

Integer-1 and integer-2 are positive, and integer-1 cannot exceed the maximum number of lines specified per report page. Integer-1 indicates an absolute line number which sets the line counter to this value after producing the last line of the current report group.

PLUS integer-2 specifies a relative line number which increments the line counter by the integer-2 value. Integer-2 represents the number of lines skipped following the last line of the current report group. Further spacing is specified by the LINE NUMBER clause of the next report group produced.

The NEXT PAGE phrase indicates an automatic skip to the next page following the last line of the current report group. Page heading and page footing report groups are produced as specified. When specified for a control footing or heading report group, NEXT GROUP results in automatic line spacing only when a control break occurs on the level for which the control footing or heading is specified. NEXT PAGE is illegal in page heading and page footing report groups.

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.

5.3.7 PAGE LIMIT

This clause indicates the specific line control to be maintained within the logical presentation of a page. PAGE LIMIT is required when page format is to be controlled by the Report Writer; it may be omitted when no association is necessary between report groups and the physical format of the report page. If PAGE LIMIT is omitted, absolute LINE NUMBER and absolute NEXT GROUP clauses must also be omitted.

Only one PAGE LIMIT clause can be specified for each RD entry.

$$\underline{\text{PAGE}}\left[\begin{Bmatrix}\underline{\text{LIMIT}}\text{ IS}\\ \underline{\text{LIMITS}}\text{ ARE}\end{Bmatrix}\right]\text{integer-1}\begin{Bmatrix}\underline{\text{LINE}}\\ \underline{\text{LINES}}\end{Bmatrix}\text{ [, }\underline{\text{HEADING}}\text{ integer-2]}$$

$$\text{[, }\underline{\text{FIRST}}\text{ }\underline{\text{DETAIL}}\text{ integer-3] [, }\underline{\text{LAST}}\text{ }\underline{\text{DETAIL}}\text{ integer-4]}$$

$$\text{[, }\underline{\text{FOOTING}}\text{ integer-5]}$$

Integer-1 through integer-5 must be positive. Integer-2 through integer-5 each must be less than or equal to integer-1.

Integer-1 is required to specify the depth of the report page; it need not be equal to the physical perforated continuous form often associated in a report with the page length.† The size of the fixed data name, LINE-COUNTER, is based on the integer-1 LINES specification. It is the maximum numeric size required to prevent overflow.

If absolute line spacing is indicated for all report groups, integer-1 must be specified, and none of the integer-2 through integer-5 controls need be specified.

If relative spacing is indicated for individual type detail report group entries, the following must be specified:

Integer-3 if a page heading is present

Integer-4 if a control heading is present

Integer-5 if a page footing is present

HEADING integer-2: First line number of first heading report group. No report group will start before integer-2.

FIRST DETAIL integer-3: First line number of first normal (detail or control) report group. No detail or control report group will start before integer-3.

† The only carriage control tape requirement is that the TOP-OF-FORM punch must be equal to LINE 0 of the report.

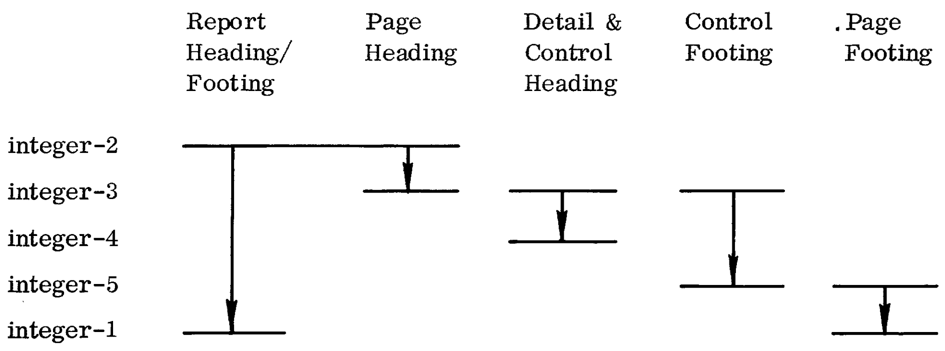
LAST DETAIL integer-4: Last line number of last normal (detail or control) report group. No detail or control heading report group will extend beyond integer-4.

FOOTING integer-5: Last line number of last control footing report group. No control footing report group will start before integer-3 nor extend beyond integer-5. Page footing report groups follow integer-5.

When relative line numbers are specified for report groups, PAGE LIMITS integer-1 is specified and some or all of HEADING, FIRST DETAIL, LAST DETAIL, and FOOTING, clauses are omitted, the following implicit control is assumed for the omitted specifications:

> If HEADING integer-2 is omitted, integer-2 is considered equivalent to the value 1, that is, LINE NUMBER one.
>
> If FIRST DETAIL integer-3 is omitted, integer-3 is considered equivalent to the value of integer-2.
>
> If LAST DETAIL integer-4 is omitted, integer-4 is considered equivalent to the value of integer-5.
>
> If FOOTING integer-5 is omitted, integer-5 is considered to be equivalent to the value of integer-4. If both LAST DETAIL integer-4 and FOOTING integer-5 are omitted, both are considered to be equivalent to the value of integer-1.

The following chart represents the limits of page format when all options of the PAGE LIMIT clause are specified:



Absolute LINE NUMBER or absolute NEXT GROUP spacing must be consistent with controls specified in the PAGE LIMIT clause.

Examples:

1.
```
REPORT SECTION.
RD   RATIO-REPORT
     PAGE LIMIT IS 55 LINES.
```

In this example, an absolute page limit is indicated.

2.
```
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.
```

In this example, automatic line control of heading, footing, and detail groups within the report page are indicated.

5.3.8 RESET

The RESET clause is used in an elementary item description in a control footing report group to override the automatic resetting of the sum counter following the associated control break.

$$\underline{\text{RESET}}\ \text{ON} \left\{ \begin{array}{l} \text{identifier-1} \\ \underline{\text{FINAL}} \end{array} \right\}$$

RESET can be used only at the elementary level in conjunction with the SUM clause (see SUM clause description). Identifier-1 must be one of the identifiers specified in the CONTROL clause of the RD entry for the report. It must be a higher level identifier than the control identifier associated with the control footing or detail report group containing the RESET clause.

When RESET is not specified, the sum counters associated with the report group by the SUM clause are automatically reset to zero after the control footing or the detail report group is presented. RESET prevents the automatic resetting of the sum counters until the control footing or detail report group associated with identifier-1 has been presented. This clause, therefore, 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 or detail report group has been generated. With this option cumulative totals are 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.

5.3.9
SOURCE - SUM - VALUE

The SOURCE, SUM, or VALUE clauses define the purpose of an elementary item within the report group; only one clause is included in the description of any one item, and it must only appear at the elementary level.

$$\left\{\begin{array}{l}\text{; }\underline{\text{SOURCE}}\text{ IS identifier-1}\\ \text{; }\underline{\text{SUM}}\text{ identifier-2 [, identifier-3] ... [}\underline{\text{UPON}}\text{ data-name-1]}\\ \text{; }\underline{\text{VALUE}}\text{ IS literal-1}\end{array}\right\}$$

The identifiers must identify an item in the File, Working-Storage, or Common-Storage Sections; a sum counter in the Report Section; or one of the special registers: SYSTEM-DATE, or SYSTEM-TIME. The literal in the VALUE clause may be numeric, non-numeric, or a figurative constant. Data-name-1 is the name of a detail report group; it may be qualified by report name if the name is used in more than one report description. SOURCE and SUM are described below; VALUE is described in chapter 3.

The SOURCE clause indicates a data item used as a source for the report item. PICTURE must also be specified in the entry for the report item. The value of identifier-1 at object time is effectively moved to the report item and presented according to the PICTURE specified in the report item description. The fixed data items LINE-COUNTER or PAGE-COUNTER may be specified as identifier-1; in this case the current value of the line counter or the page counter is the source.

Examples:

```
1.  01  DETAIL-ITEM TYPE DETAIL LINE PLUS 1.
        02  DEPT PIC X COLUMN 70 SOURCE IS DEPT-NO.
```

In this example, DEPT-NO is the name of a data item in the File, Working-Storage, or Common-Storage Section.

```
2.  01  TYPE IS PAGE HEADING LINE PLUS 02.
        02  COLUMN 110 PIC X(4) VALUE IS "PAGE".
        02  COLUMN 115 PIC 999 SOURCE IS PAGE-COUNTER.
```

The SUM clause indicates values to be accumulated when a control break occurs at object time. It may appear only in a control footing or a detail report group at the elementary level. Any item containing a SUM clause generates a numeric counter used for summing the operands specified by the identifiers following the word SUM. This summation counter can be referenced by specifying the data name of the item containing the SUM clause. If a summation counter is never referenced, a data name need not be included in the entry containing SUM. A PICTURE clause must always be included in an entry containing a SUM clause. Editing characters or editing clauses may be included but editing occurs only upon presentation of the summation counter in the report. At all other times the summation counter is treated as a numeric data item. The PICTURE must specify a size large enough to accommodate the summed quantity without truncation of integral digits.

Each item being summed (identifier-3, identifier-4, etc.) must appear as the object of a SOURCE clause in a detail report group, name an entry containing a SUM clause in a control footing report group at an equal or lower position in the control hierarchy, or name an entry containing a SUM clause in a detail report group of the lowest level of the control hierarchy. The items being summed must be explicitly included in a detail report group, but they may be suppressed at presentation time.

The summation of data items defined as summation counters in control footing or detail report groups is accomplished explicitly or implicitly by the Report Writer. A summation counter is algebraically incremented just before the detail report group which contains the data items being summed is presented.

The items being summed (identifier-3, identifier-4, etc.) are added to the summation counter at each execution of a GENERATE statement. This statement generates a detail report group that contains the SUM operands at the elementary level. Summing is done according to the rules for ADD in the Procedure Division (section 4.7.2).

If the sum of a data item is to be presented at a higher level of the control hierarchy, the lower level SUM specification may be omitted. If higher level report groups are indicated in the control heirarchy, counter updating procedures take place prior to the reset operation. Unless RESET is specified, each summation counter at the level just produced is reset to zero.

The UPON data-name-1 option provides selective summation when a particular data item is named in the SOURCE clause of two or more detail report groups. Identifier-3, identifier-4, etc., must be named in a SOURCE clause in the detail report group identified by data-name-2.

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.
```

If AMNT-NO is the object of a SOURCE clause in a detail report group, each time the detail group is presented the contents of AMNT-NO will be added to the summation counter associated with AMNT-NO. The contents of this counter will be presented when the control footing report group is presented.

5.3.10
TYPE

This clause specifies the particular type of report group that is described by this entry and indicates the time at which the report group is to be generated. It must appear in an 01 level entry.

```
          ⎧ {REPORT HEADING}                                  ⎫
          ⎪ {RH            }                                  ⎪
          ⎪ {PAGE HEADING}                                    ⎪
          ⎪ {PH          }                                    ⎪
          ⎪ {CONTROL HEADING} {identifier-n}                  ⎪
          ⎪ {CH             } {FINAL       }                  ⎪
TYPE IS   ⎨ {DETAIL}                                          ⎬
          ⎪ {DE    }                                          ⎪
          ⎪ {CONTROL FOOTING} {identifier-n}                  ⎪
          ⎪ {CF             } {FINAL       }                  ⎪
          ⎪ {PAGE FOOTING}                                    ⎪
          ⎪ {PF          }                                    ⎪
          ⎪ {REPORT FOOTING}                                  ⎪
          ⎩ {RF            }                                  ⎭
```

The level number 01 identifies the particular report group to be generated as output. The TYPE clause indicates the time for generating this report group. A report group described as other than DETAIL is automatically generated by the Report Writer. If the report group is described as DETAIL, the Procedure Division statement, GENERATE identifier, directs the Report Writer to produce the named report group.

The REPORT HEADING or RH entry indicates a report group that is produced only once at the beginning of a report during execution of the first GENERATE statement. Only one report group of this type can appear in a report. Nothing may precede a REPORT HEADING entry in a report. SOURCE clauses used in TYPE RH report groups refer to the values of data items at the time the first GENERATE statement is executed.

The PAGE HEADING or PH entry indicates a report group produced at the beginning of each page according to page condition rules. Only one report group of this type can appear in a report.

The CONTROL HEADING or CH entry indicates a report group produced at the beginning of a control group for a designated identifier. CH FINAL indicates a report group produced once before the first control group at the initiation of a report during execution of the first GENERATE statement. Only one report group of this type can appear for each identifier and for the FINAL specified in a report. To produce CONTROL HEADING report groups, a control break must occur (see section 5.1.1). SOURCE clauses used in TYPE CONTROL HEADING FINAL report groups refer to the values of the items at the time the first GENERATE statement is executed.

The DETAIL or DE entry indicates a report group produced for each GENERATE statement in the Procedure Division. Each DETAIL report group must have a unique data name at the 01 level in a report.

The CONTROL FOOTING or CF entry indicates a report group produced at the end of a control group for a designated identifier or produced once at the termination of a report ending a FINAL control group. Only one report group of this type can appear for each identifier and for the FINAL specified in a report. To produce any CONTROL FOOTING report groups, a control break must occur.

SOURCE clauses in TYPE CONTROL FOOTING FINAL report groups refer to the values of the items at the time the TERMINATE statement is executed.

The PAGE FOOTING or PF entry indicates a report group produced at the bottom of each page. Only one report group of this type can appear in a report.

The REPORT FOOTING or RF entry indicates a report group produced only once at the termination of a report. Only one report group of this type can be used in a report. Nothing may follow a REPORT FOOTING entry in a report. SOURCE clauses in REPORT FOOTING report groups refer to the value of items at the time the TERMINATE statement is executed.

CONTROL HEADING report groups appear with the current values of any indicated SOURCE data items before the DETAIL report groups of the control group are produced. CONTROL FOOTING report groups appear with the previous values of any indicated SOURCE data items just after the DETAIL report groups of that control group have been produced. These report groups appear when a control break is noted. LINE NUMBER determines the absolute or relative position of the CONTROL report groups exclusive of the other HEADING and FOOTING report groups.

Identifier-n, as well as FINAL, must be one of the identifiers described in the CONTROL clause of the RD entry. A FINAL control break may be designated only once for CONTROL HEADING or CONTROL FOOTING entries within a report.

HEADING and FOOTING report groups occur in the following sequence if all exist for a given report:

REPORT HEADING (one occurence only)
PAGE HEADING
:
CONTROL HEADING
DETAIL
CONTROL FOOTING
:
PAGE FOOTING
REPORT FOOTING (one occurrence 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

5.4 PROCEDURE DIVISION STATEMENTS

5.4.1 GENERATE

The GENERATE statement links the Procedure Division to the Report Writer as described in the Report Section of the Data Division.

GENERATE identifier

Identifier represents a detail report group or an RD entry.

If identifier is the name of a detail report group, the GENERATE statement produces all the automatic operations within a Report Writer and produces an output detail report group on the output device. This procedure is called detail reporting.

If identifier is the name of an RD entry, the GENERATE statement produces all the automatic operations of the Report Writer and updates the footing report groups within a particular report description without actually producing a detail report group associated with the report. In this case, all summation counters associated with the report description are algebraically incremented. This procedure is called summary reporting. If more than one detail report group is specified in a report defined by the RD entry, all summation counters are algebraically incremented each time a GENERATE statement is executed.

A GENERATE statement, implicitly in both detail and summary reporting, produces the following automatic operations:

- Recognizes specified control breaks to produce control footing and control heading report groups
- Accumulates into the summation counters all specified identifiers
- Executes specified routines defined by a USE statement before generating associated report groups (See USE, section 5.4.4)
- Steps and tests the line counter and page counter to produce page footing and page heading report groups
- Creates and prints the print line image
- Resets the summation counters unless suppressed by the RESET clause

During execution of the first GENERATE statement, report groups are produced in the following order:

REPORT HEADING report group

PAGE HEADING report group

all CONTROL HEADING report groups in the order FINAL, major to minor

DETAIL report group, if specified in the GENERATE statement

If a control break is recognized when any GENERATE statement after the first is executed, all specified control footing report groups are produced from the minor report group up to and including the report group specified for the identifier which caused the control break. Then, all specified control heading report groups from the report group specified for the identifier that caused the control break down to the minor report group, are produced in that order. The detail report group specified in the generate statement is then produced.

Data is moved to the data item in the report group description entry of the Report Section, and it is edited under control of the Report Writer according to the same rules for movement and editing as described for MOVE.

5.4.2 INITIATE

INITIATE

Report processing begins with the INITIATE statement.

INITIATE report-name-1 [, report-name-2]...

Each report name must be defined by a report description entry in the Report Section of the Data Division.

The INITIATE statement resets all data name entries that contain SUM clauses associated with the report; and it sets up the Report Writer controls for all report groups associated with this report.

If PAGE-COUNTER is specified, it is set to one before or during execution of the INITIATE statement. If the starting value is to be other than one, the user may reset this counter following the INITIATE statement.

If LINE-COUNTER is specified, it is set to zero before or during the execution of INITIATE.

The INITIATE statement performs Report Writer functions for individually described report programs analogous to the input-output functions that the OPEN statement performs for individually described files.

A second INITIATE statement for a particular report name may not be executed unless an intervening TERMINATE statement has been executed for that report name.

5.4.3 TERMINATE

Report processing is completed by the TERMINATE statement.

```
    TERMINATE report-name-1 [, report-name-2]...
```

Each report name must be defined by an RD entry in the Data Division.

The TERMINATE statement produces all control footings associated with this report as if a control break had just occurred at the highest level; and it completes the Report Writer functions for the named reports. TERMINATE also produces the last page and report footing report groups associated with this report. Page heading or page footing report groups are prepared in order for the report description.

A second TERMINATE statement for a particular report may not be executed unless an intervening INITIATE statement has been executed for the report name.

The TERMINATE statement performs Report Writer functions for individually described report programs analogous to the input-output functions that the CLOSE statement performs for individually described files. TERMINATE does not close the file with which the report is associated; a CLOSE statement for the file must be given by the user.

SOURCE clauses used in final control footing or report footing report groups refer to the values of the items during execution of the TERMINATE statement.

5.4.4 USE BEFORE REPORTING

The USE statement specifies Procedure Division statements to be executed just before a report group is produced.

<u>USE</u> <u>BEFORE</u> <u>REPORTING</u> identifier-1 [, identifier-2]...

A USE statement is specified immediately after a section header in the Declarative portion of the Procedure Division; it must be followed by a space. The remainder of the section must consist of one or more procedural paragraphs that define the procedures to be used.

Each identifier represents a report group named in the Report Section of the Data Division. An identifier must not appear in more than one USE statement.

No Report Writer statement (GENERATE, INITIATE, or TERMINATE) may be written in a procedural paragraph following the USE sentence in the declarative portion.

The USE statement itself is never executed, rather it defines the conditions calling for the execution of the USE procedures. The designated procedures are executed by the Report Writer just before the named report is produced, regardless of page or control break associations with report groups. If USE procedures are specified for a CONTROL FOOTING report group, they affect the previous value of items specified in the CONTROL clause or the current value of items specified in a SOURCE clause.

A USE procedure must not contain any reference to non-declarative procedures. Conversely, the non-declarative portion must not contain reference to procedure-names that appear in the declarative portion, except that PERFORM statements may refer to a USE declarative or to the procedures associated with the USE declarative.

```
JOB ACCOUNTING INFORMATION

  NAME=VIANDS       ACCT=NLVK3LW1

  DATE=10/14/68     EDITION=UC       TIME-ON=13/16/57   TIME-OFF=13/17/34

TIME USED
  COMP=00/00/19.354
  CHAN=00/00/08.642

FACILITIES NOT USED
  CORE=018
  SCR =006
  LINE=3702
  CARD=499

JOB,NLVK3LW1,VIANDS,60,5000,500
SCHED,CORE=50,SCR=15,CLASS=1,ABORT=1500
UCBL(L,M,O,X)




 USASI COBOL 1.0 / MASTER 2.1           REPORT-WRITER-EXAMPLE                      10/14/68

      00001                   IDENTIFICATION DIVISION.
      00002                   PROGRAM-ID. REPORT-WRITER-EXAMPLE.
      00003                   AUTHOR. ANONYMOUS Q CITIZEN.
      00004                   DATE-WRITTEN. 02/09/68.
      00005                   DATE-COMPILED.   10/14/68
      00006                   ENVIRONMENT DIVISION.
      00007                   CONFIGURATION SECTION.
      00008                   SOURCE-COMPUTER. 3300.
      00009                   OBJECT-COMPUTER. 3300.
      00010                   INPUT-OUTPUT SECTION.
      00011                   FILE-CONTROL.
      00012                        SELECT FILE-A ASSIGN TO SYSTEM-INPUT.
      00013                        SELECT FILE-B ASSIGN TO SYSTEM-OUTPUT.
```

```
                       DATA DIVISION.
                       FILE SECTION.
                       FD  FILE-A
                               LABEL RECORDS ARE OMITTED
                               DATA RECORD IS CARD-IMAGE.
         D00010        01  CARD-IMAGE.
         D00011            02 DEPT-NO PICTURE X(5).
         D00012            02 SECT-NO PICTURE X(5).
         D00013            02 GROUP-NO PIC X(5).
         D00014            02 AMNT-NO PIC 9(3)V9(2).
                       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 50
                               FOOTING 55.
                       01  TYPE IS REPORT HEADING LINE 02.
                           02  COLUMN 02 PIC XX VALUE IS ≠RH≠.
                           02  COLUMN 52 PIC X(15) VALUE ≠EXPENSE ACCOUNT≠.
                       01  TYPE IS PAGE HEADING LINE PLUS 2.
                           02  COLUMN 3 PIC XX VALUE ≠PH≠
                           02  COLUMN 110 PIC X(4) VALUE ≠PAGE≠.
                           02  COLUMN 115 PIC 999 SOURCE IS PAGE-COUNTER.
                           02  LINE PLUS 1 COLUMN 90 PIC X(8) VALUE ≠GROUP NO≠.
                           02  COLUMN 100 PIC X(7) VALUE ≠EXPENSE≠.
                       01  TYPE IS CONTROL HEADING FINAL LINE PLUS 2.
                           02  COLUMN 4 PIC XXX VALUE ≠CHF≠.
                           02  COLUMN 60 PIC X(17) VALUE ≠ITEMIZED BY GROUP≠.
                       01  TYPE IS CONTROL HEADING DEPT-NO LINE PLUS 2.
                           02  COLUMN 5 PIC XXX VALUE ≠CHA≠.
                           02  COLUMN 70 PIC X(7) VALUE ≠DEPT-NO≠.
                       01  TYPE IS CONTROL HEADING SECT-NO LINE PLUS 1
                                   NEXT GROUP PLUS 2.
                           02  COLUMN 6 PIC XXX VALUE ≠CHB≠.
                           02  COLUMN 80 PIC X(7) VALUE ≠SECTION≠.
                       01  DETAIL-ITEM TYPE DETAIL LINE PLUS 1.
                           02  COLUMN 7 PIC XX VALUE ≠DE≠.
                           02  DEPT PIC X(5) COLUMN 70 SOURCE DEPT-NO GROUP INDICATE.
```

```
USASI COBOL 1.0 / MASTER 2.1          REPORT-WRITER-EXAMPLE               10/14/68

                         02  SECT PIC X(5) COLUMN 80 SOURCE SECT-NO GROUP INDICATE.
                         02  GRP-N PIC X(5) COLUMN 90 SOURCE GROUP-NO.
                         02  AMOUNT PIC ZZZ.ZZ COLUMN 100 SOURCE AMNT-NO.
                     01  TYPE IS CONTROL FOOTING SECT-NO LINE PLUS 2.
                         02  COLUMN 6 PIC XXX VALUE ≠CFB≠.
                         02  COLUMN 89 PIC X(10) VALUE ≠SEC TOTAL≠.
                         02 COLUMN 100 PIC ZZZZZZ.ZZ SUM AMNT-NO.
                     01  TYPE IS CONTROL FOOTING DEPT-NO LINE PLUS 2.
                         02  COLUMN 5 PIC XXX VALUE ≠CFA≠.
                         02  COLUMN 70 PIC X(11) VALUE ≠DEPT. TOTAL≠.
                         02 COLUMN 100 PIC ZZZZZZ.ZZ SUM AMNT-NO.
                     01  TYPE IS CONTROL FOOTING FINAL LINE PLUS 2.
                         02  COLUMN 4 PIC XXX VALUE ≠CFF≠.
                         02  COLUMN 60 PIC X(11) VALUE ≠GRAND TOTAL≠.
                         02 COLUMN 100 PIC ZZZZZZ.ZZ SUM AMNT-NO.
                     01  TYPE IS PAGE FOOTING LINE PLUS 2.
                         02  COLUMN 3 PIC XX VALUE ≠PF≠.
                         02  COLUMN 110 PIC X(4) VALUE ≠PAGE≠.
                         02  COLUMN 115 PIC 999 SOURCE IS PAGE-COUNTER.
                     01  TYPE IS REPORT FOOTING LINE PLUS 2.
                         02  COLUMN 2 PIC XX VALUE ≠RF≠.
                         02  COLUMN 30 PIC X(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.
                     END PROGRAM.
```

```
RH                        EXPENSE ACCOUNT
 PH                                                                             PAGE 001
                                                     GROUP NO  EXPENSE
  CHF                          ITEMIZED BY GROUP
   CHA                                  DEPT-NO
    CHB                                           SECTION
     DE                                 A0010     AA001     AAA01     100.50
     DE                                                     AAA02     120.50
     DE                                                     AAA03     135.00
     DE                                                     AAA04     125.90
     DE                                                     AAA05     158.20
    CFB                                                    SEC TOTAL     640.10
    CHB                                           SECTION
     DE                                 A0010     AA002     AAA01       2.50
     DE                                                     AAA02      28.50
     DE                                                     AAA50     356.00
     DE                                                     AAA60     505.00
    CFB                                                    SEC TOTAL     892.00
    CHB                                           SECTION
     DE                                 A0010     AA050     AAA01     600.00
    CFB                                                    SEC TOTAL     600.00
    CHB                                           SECTION
     DE                                 A0010     AA060     AAA30     785.00
    CFB                                                    SEC TOTAL     785.00
   CFA                                  DEPT. TOTAL                     2917.10
   CHA                                  DEPT-NO
    CHB                                           SECTION
     DE                                 A0020     AA003     AAA06     500.50
     DE                                                     AAA07     655.00
    CFB                                                    SEC TOTAL    1155.50
 PF                                                                             PAGE 001
```

```
PH                                                                         PAGE 002
                                                     GROUP NO  EXPENSE
  CHB                                        SECTION
   DE                               A0020    AA004    AAA16    425.00
  CFB                                                SEC TOTAL     425.00
  CFA                               DEPT. TOTAL                   1580.50
  CHA                               DEPT-NO
   CHB                                       SECTION
    DE                              A0030    AA005    AAA17    525.25
   CFB                                               SEC TOTAL     525.25
  CFA                               DEPT. TOTAL                    525.25
  CHA                               DEPT-NO
   CHB                                       SECTION
    DE                              A0040    AA006    AAA20    682.00
   CFB                                               SEC TOTAL     682.00
   CHB                                       SECTION
    DE                              A0040    AA007    AAA30    725.00
   CFB                                               SEC TOTAL     725.00
  CFA                               DEPT. TOTAL                   1407.00
  CHA                               DEPT-NO
   CHB                                       SECTION
    DE                              A0050    AA080    AAA40    605.80
   CFB                                               SEC TOTAL     605.80
  CFA                               DEPT. TOTAL                    605.80
 PF                                                                        PAGE 002
```

```
PH                                                                PAGE 003
                                           GROUP NO  EXPENSE
  CHA                     DEPT-NO
   CHB                              SECTION
    DE                    A0060     AA100     AAA50     725.00

   CFB                                       SEC TOTAL     725.00
   CHB                              SECTION
    DE                    A0060     AA150     AAA60     688.85

   CFB                                       SEC TOTAL     688.85

  CFA                     DEPT. TOTAL                     1413.85

  CHA                     DEPT-NO
   CHB                              SECTION
    DE                    A0070     AA250     AAA70     101.05

   CFB                                       SEC TOTAL     101.05
   CHB                              SECTION
    DE                    A0070     AA260     AAA80     189.50

   CFB                                       SEC TOTAL     189.50

  CFA                     DEPT. TOTAL                      290.55

  CHA                     DEPT-NO
   CHB                              SECTION
    DE                    A0080     AA300     AAA90     198.50

   CFB                                       SEC. TOTAL    198.50

  CFA                     DEPT. TOTAL                      198.50
PF                                                                PAGE 003
```

```
PH                                                                      PAGE 004
                                                         GROUP NO  EXPENSE



  CHA                                        DEPT-NO
   CHB                                                  SECTION

    DE                                       A0090      AA350      AAA95         .50

   CFB                                                          SEC TOTAL          .50

  CFA                                        DEPT. TOTAL                           .50

  CHA                                        DEPT-NO
   CHB                                                  SECTION

    DE                                       A0100      AA380      AAA99     995.00

   CFB                                                          SEC TOTAL       995.00

  CFA                                        DEPT. TOTAL                        995.00

 CFF                                  GRAND TOTAL                              9934.05




  PF                                                                    PAGE 004

 RF                    THIS COMPLETES THE MONTHLY EXPENSE REPORT
```

LIBRARY 6

The COBOL library contains text that is available to a source program at compile time. Compilation of library text is effectively the same as if the text were actually written as part of the source program.

The COBOL library may contain text for the Environment Division, the Data Division, and the Procedure Division. Library text is made available through the COPY statement.

6.1 COPY STATEMENT

```
COPY library-name

┌ REPLACING {word-1      }  BY  {word-2      }            ┐
│           {identifier-1}      {identifier-2}            │
│                                                         │
│         [ , {word-3      }  BY  {word-4      } ]  ...   │
└         [   {identifier-3}      {identifier-4} ]        ┘
```

A word is any COBOL word in a library routine that is not on the list of COBOL reserved words (appendix C).

The COPY statement may appear:

- In any Environment Division paragraphs
- In any FD, SD, RD or 01 level entry in the Data Division
- In a Procedure Division section or paragraph

No other statement or clause may appear in the same entry as the COPY statement.

The library text is copied from the library at compilation time. The result is the same as if the text were actually part of the source program. The COPY process is terminated by the end of the library text.

If the REPLACING option is used, each word or identifier specified in the format is replaced by a corresponding word or identifier when COPY is executed. Replacement of one identifier by another includes all associated qualifiers, subscripts, and indexes. Use of the REPLACING option does not alter the text as it appears in the library.

The library must not contain any COPY statements.

The COPY statement may be written as follows:

The COPY statement may be written as follows:

ENVIRONMENT DIVISION.

$$\left\{\begin{array}{l}\underline{\text{SOURCE-COMPUTER.}}\\ \underline{\text{OBJECT-COMPUTER.}}\\ \underline{\text{SPECIAL-NAMES.}}\\ \underline{\text{FILE-CONTROL.}}\\ \underline{\text{I-O-CONTROL.}}\end{array}\right\}\quad \underline{\text{COPY}}\text{ statement.}$$

DATA DIVISION

FILE SECTION

$$\left\{\begin{array}{l}\underline{\text{FD}}\text{ file-name}\\ \underline{\text{SD}}\text{ sort-file-name}\\ \text{01 data-name}\end{array}\right\}\quad \underline{\text{COPY}}\text{ statement.}$$

WORKING-STORAGE SECTION

01 data-name <u>COPY</u> statement.

REPORT SECTION.

$$\left\{\begin{array}{l}\underline{\text{RD}}\text{ report-name}\\ \text{01 data-name}\end{array}\right\}\quad \underline{\text{COPY}}\text{ statement.}$$

PROCEDURE DIVISION

procedure-name. <u>COPY</u> statement

6.2 SOURCE LIBRARY PREPARATION

The information to be copied may appear on the system library file (*LIB) or on an auxiliary library file. These are mass storage files referenced through the system library directory (*DIR) or an auxiliary library directory. COBOL source library entries are placed on a library file using the MASTER library generation routine GLIB. GLIB is described in the MASTER Installation Manual. BCD COBOL source statements are written on the library in card image format and an entry is placed in the non-resident library subprogram directory using the $BCD card as follows:

```
$BCD, library-name
COBOL source statements
```

Each Environment or Procedure Division paragraph, file description, record description, or report description entry must be placed on the library as a separate $BCD entry.

Library names may not exceed eight characters.

Examples:

```
$BCD,SCOMP
        SOURCE-COMPUTER,3300.

$BCD,PAY-FILE
        FD PAY-FILE LABEL RECORD OMITTED
        DATA RECORD IS PAY-CARD.

$BCD,PAY-CARD
        01  PAY-CARD
                    02  NAME PIC X(30).
                    02  PAY-RATE PIC 9(5).

$BCD,SECT-1
        MASTER-UPDATE SECTION.
        PARAGRAPH-1.
            OPEN INPUT... .
              .
              .
              .

$BCD,PAR-2
        PARAGRAPH-2.
            CLOSE ... .
              .
              .
              .
```

The $BCD card always begins in card column 1; the COBOL source statements follow the COBOL coding sheet format (chapter 8).

SOURCE PROGRAM PREPARATION, COMPILATION, EXECUTION 7

7.1 REFERENCE FORMAT

The reference format is the standard method for describing COBOL source programs. It is described in terms of character positions in 80-character source records read from the standard input file (INP) or from the dsi supplied by the user in the UCBL control card parameter, I = dsi.

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

Each division must be written according to the reference format rules; and the divisions must be ordered as follows:

Identification Division

Environment Division

Data Division

Procedure Division

Each source record is equivalent to one line of the COBOL coding sheet. This source line is divided into five areas as follows:

Character Position	Area
1-6	Sequence Number
7	Continuation
8-11	Area A
12-72	Area B
73-80	Identification

7.1.1 COBOL CODING SHEET

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 area A 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. The remainder of the line is blank, except if the section is a DECLARATIVE, it may be followed by a USE or COPY sentence. Paragraph names are followed by a period and at least one space. The text may follow or may start in area B of the next line.

The level indicators: FD, SD, RD, the level numbers: 01 and 77 begin in area A. They are followed by one or more spaces and the associated entry. All other level numbers begin in area B followed by a space and associated entry.

Sequence numbers, if specified, are in the sequence number area. Program identification is placed in the identification code area. 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, (continuation area). If a non-numeric literal is split between two lines, a quotation mark must be specified in area B 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.

COBOL Coding Sheet Rules

Element	Type	Division	Reference Area	Remarks
Name	Division-name	ALL	Area A	Name must be followed by a period; remainder of the line must be blank.
	Section-name	ENVIRONMENT DATA PROCEDURE	Area A	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	Area A	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	Area A	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 Report Element Description	DATA	Area A or Area B	Same as file description entry. Level number 01-49, 66, 77 and 88. Only 01,77 entries may begin in Area A.
Sentence	First sentence of a paragraph or section	IDENTIFICATION ENVIRONMENT PROCEDURE	Following period and 1 space after paragraph or section name, or on next line in Area B.	
	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	Area B	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	Area B	
Non-Program Entry	Sequence Number	ALL	Sequence Area	Sequence number does not affect the object program; processor does check for correct sequencing.
	Program Identification	ALL	Identification Area	Identification information does not affect object program.

7.2 COBOL CONTROL CARDS

7.2.1 UCBL CARD

This card, which signals MASTER to call the COBOL compiler, is placed directly before the Identification Division of the source deck. It may contain up to nine parameters which specify input/output options provided by the compiler.

$UCBL ($p_1$,$p_2$,...$p_9$)

Parameters are free field, separated by commas; they have the general form:

option = dsi

The option must begin with a character I, P, X, L, M, C, O, or T.

Additional characters preceding the equal sign are ignored; for example, L and LIST are the same parameter. If only the option is stated, COBOL will make a standard assignment for the option. The user is responsible for opening the file and giving it the proper dsi before calling COBOL.

INPUT = dsi	Source input file; if the parameter or file equation is omitted, the standard input file, INP, is assumed.
PUNCH = dsi	Punch; dsi represents a data set identifier assigned to an output file or device. If the parameter is absent, no punch output is produced. If only P appears, binary output is produced on the standard punch file, PUN.
XECUTE = dsi	Binary output for load-and-go; dsi represents a data set identifier assigned to a read/write file. If the parameter is absent, no load-and-go file will be written. If only X appears, binary output will be produced on the standard load-and-go file, LGO.
LIST = dsi	Source code list; dsi represents a data set identifier assigned to an output file or device. If the parameter is absent, no listing will be produced. If only L appears, the listing will be produced on the standard output file, OUT.
MAP	Supplies a memory map in the Data Division only if L is also specified; the map is produced on the same dsi as the list.

OBJECT	Provides a listing of the object code generated for the program. A mnemonic operation code and data reference symbols as well as an octal representation are produced for each instruction. This option may be specified only when LIST is specified. The object is produced on the same dsi as the list.
COPY = fdsi/ddsi	Identifies library files; fdsi is the data set identifier of the file from which source COBOL statements are copied when COPY is specified in the source program. ddsi is the data set identifier of the directory used to locate the source statement entries on the library file. If C appears alone or is omitted, *LIB/*DIR is used when a COPY statement is encountered.
TRACE	Indicates embedded TRACE statements are to be compiled; if omitted, all TRACE statements within the source program are ignored during compilation.

7.2.2 END PROGRAM CARD

This card indicates the end of the source program to the COBOL compiler. It contains the specification END PROGRAM beginning in column 8, and should follow the last card in the Procedure Division of the source deck.

7.2.3 FINIS CARD

The FINIS card signals the end of compilation and returns control to MASTER. It consists of the word FINIS starting in or to the right of column 8.

If only one COBOL program is submitted for compilation, the FINIS card is placed behind the END PROGRAM card. If more than one COBOL program is submitted for compilation, the END PROGRAM card for the first program is followed by the Identification Division for the second program, and so on. The FINIS card is placed after the last END PROGRAM card. FINIS is correctly recognized only after an END PROGRAM card.

7.2.4 ENDATA CARD

The ENDATA card signals that the end of data has been reached on the system input file (INP) or the input file on the card reader. It consists of the word ENDATA beginning in column 1. The ENDATA card should be inserted directly behind the user's data deck. If the deck has MASTER control cards between the end of the data deck and the standard MASTER end-of-file card, the ENDATA card precedes the MASTER control cards.

7.3 MASTER CONTROL CARDS

MASTER control cards contain a $ punch in column one, followed by the statement name and parameters, separated by commas. Only cards required for a compile-only and a compile-and-execute run are described in this manual. A complete discussion of the options available under MASTER is found in the MASTER Reference Manual, Publication number 60213600. The UCBL card is a MASTER control card.

7.3.1 JOB CARD

All jobs submitted for processing under MASTER require a JOB card. It supplies information to the installation accounting routine, identifies the programmer, and sets a job processing time limit.

```
$JOB,c,i,t,lc,pc
```

c	Account number, 0-8 characters
i	Programmer identification, 0-8 characters
t	Job time limit in minutes for entire job maximum 1440 minutes; if blank, job time depends on the installation accounting routine
lc	line count, specifies size of standard output file
pc	punch count, specifies size of standard punch file

The c and i fields are mandatory. If one of the fields is blank, the job is terminated. The JOB card is written on OUT.

7.3.2 SCHEDULE CARD

```
$SCHED, SCR=x, CORE=x,...
```

Under MASTER $SCHED cards are optional; but at least one is required for COBOL users. Card format is free field and the entires may occur in any order. Blanks or commas may be used as separators, but are not required.

The fields required by COBOL follow:

SCR=x	1-3 decimal digits describing number of segments of system scratch required to hold the intermediate output of the COBOL compiler and the user's scratch files during execution. SCR=8 is the minimum required for compilation.

CORE=x	1-3 decimal digits describing the number of 512 word quarter pages required for the run. If object program is larger than the compiler, the user must reserve additional memory. CORE=35 is suggested as a minimum for normal compilation.

Optional fields that may be required by the object program follow:

CLASS=	E	Emergency
	B	Background
	I	Input/Output (usual class for COBOL jobs)
	C	Compute
peq=	peq is a hardware type number, decimal digits to the right of = indicate the quantity of equipment of this type required to handle user files. Drives must be reserved for all Class B files. Unit record devices must also be reserved.	

7.3.3 *DEF CARD

```
$*DEF(p1,p2,...)
```

These cards are used to allocate, open, or close user files. Complete descriptions appear in the MASTER Reference Manual. *DEF cards are not required for COBOL compilation or execution except:

User must OPEN a nonstandard file for INPUT, LIST, PUNCH, XECUTE, or COPY, and he must specify the dsi for the file on the $UCBL card.

User must allocate all permanent online mass storage files which are not in existence when execution starts. Other mass storage files are allocated internally during program execution when the programmer includes the FILE LIMITS clause in the source program. OPEN source statements in the user program open all allocated files except a rerun dump file on mass storage, this must be opened with a *DEF card prior to object time. If the file to be allocated is blocked, the user must add eight characters to the block size for an I/O control system mass storage block header.

7.3.4 TASK NAME CARDS

```
$name,dsi(p1,p2,p3,...)
```

This MASTER control card is read by the job monitor, an operating system task. The file specified by dsi is positioned at its beginning and the first task is loaded. However, if the dsi is *LIB, the library directory is searched and the named task is located before it is loaded.

On a compile and execute run, this card follows the FINIS card.

Since only the name is checked for library tasks, the task name card will usually appear as follows:

```
$name,LGO.
```

7.4 OUTPUTS FROM COMPILATION

7.4.1 SOURCE PROGRAM LISTING

The COBOL compiler provides a printer listing of the source program. The lines are exact images of the cards in the source program deck. New line numbers are generated corresponding to the card image on the listing. The sequence numbers from the input cards are also printed.

7.4.2 ERROR DIAGNOSTICS

Compiler diagnostics follow the source program listing and object code listing. Line numbers adjacent to the diagnostics serve as cross reference to the original source line. Data names appear in coded form in the diagnostics. The user can refer to the Data Division portion of the source listing where a code was generated for the data name of first encounter.

The only errors recognized by the compiler are those in which the user has broken rules of the COBOL language; it does not recognize faulty programming logic, unless this also produces a language error.

All compiler diagnostics are listed in Appendix I.

7.4.3 SYMBOLIC LISTING OF OBJECT PROGRAM

When O is specified on the $UCBL card, an object program listing is printed following the source program listing. For each instruction generated, a mnemonic operation code and data reference codes are presented. An octal representation of each instruction is also printed.

7.4.4 OBJECT PROGRAM

When X or P is specified on the COBOL control card, the object program is written on load-and-go or standard punch files. The decks produced may be executed in the usual manner.

7.4.5 DATA MAP

When M is specified in combination with L on the COBOL control card, a data map of the information described in the Data Division of the source program is printed following the source program listing.

7.5 RERUN/RESTART PROCEDURES

7.5.1 RERUN

When the RERUN option of the I-O-CONTROL paragraph is specified in the source program, special control cards are required to create a rerun dump file and to produce absolute task dumps which may later be used to restart the job. Required control cards are listed below. Cards to allocate object-time mass storage space for user files are not shown:

$JOB,...	
$SCHED,...	
$*DEF(A,...)	Allocate the rerun dump file (mass storage only); block size must be at least 256 characters
$*DEF($\begin{smallmatrix}O\\U\end{smallmatrix}$,W,rdsi,...)	Open the rerun dump file
$xxxx,idsi	Call task xxxx from idsi (INP, LGO, etc.)

If the rerun dump file is on tape, the allocate card is not required and the open card will contain U as the first parameter. Only one rerun dump file may be opened for a given job.

7.5.2 RESTART

To restart a program from one of the dumps on the rerun dump file, the following cards are required:

$JOB, . . .	
$SCHED, . . .	Must have same parameters as original
$*DEF($\overset{O}{U}$, W, rdsi, . . .)	Openthe rerun dump file
$xxxx, idsi(rdsi, n)	Task name card

xxxx	Task name
idsi	dsi of input file (INP, etc.)
(rdsi,	Restart from rdsi; parameter passed to COBOL restart routine
n)	Number of rerun dump to be used for restart; n = 1-9999. If n is omitted, the last dump on the file will be loaded. The comma must be present.

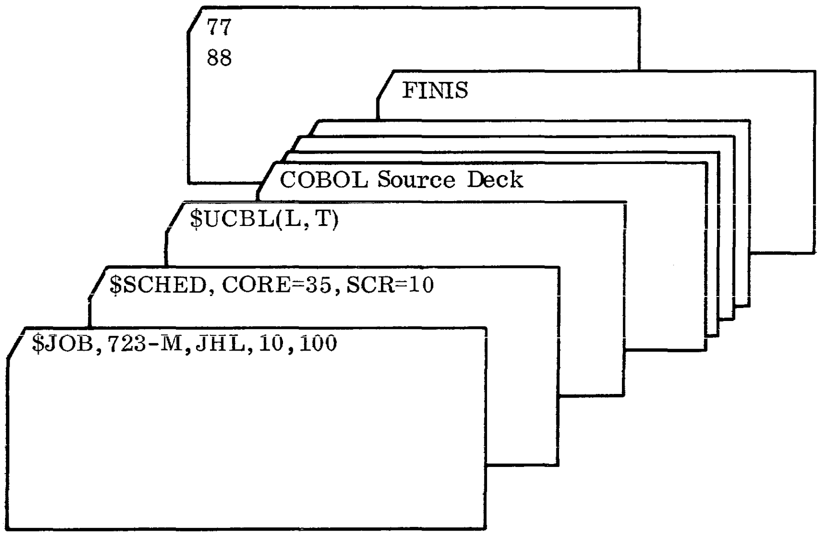
Mass storage ALLOCATE cards should not be included in a restart run of the program.

The restart routine re-positions all files to their locations at the time rerun dump was taken. The rerun dump file is poritioned so that the next dump follows the one used to restart the program. If input data came from system input (INP) the entire data deck must be reloaded into the card reader.

7.6 SAMPLE COBOL DECKS

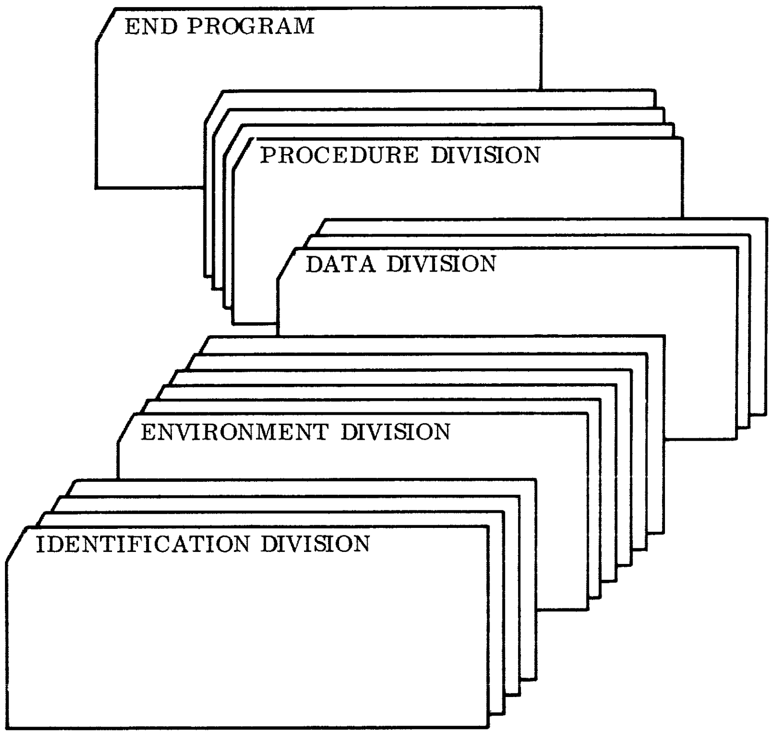
7.6.1 COMPILE ONLY

COMPILE ONLY With LIST and TRACE Options



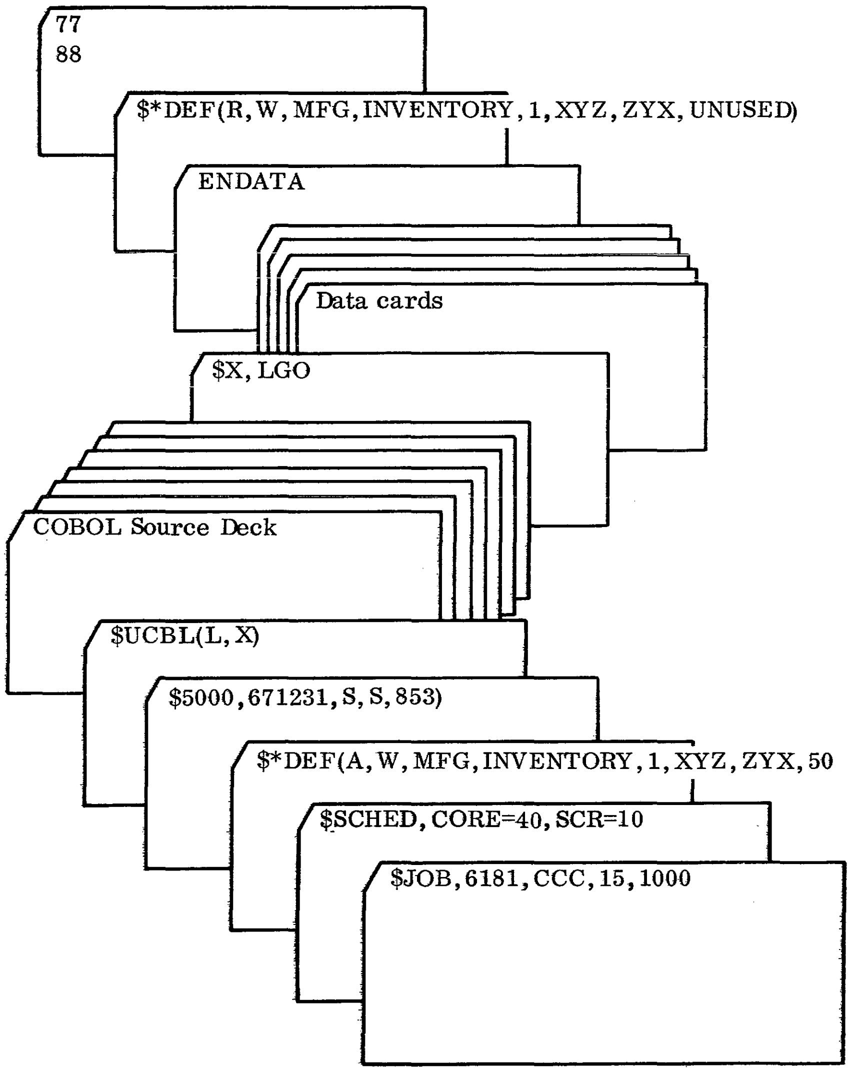
7.6.2
COBOL SOURCE DECK

COBOL SOURCE DECK



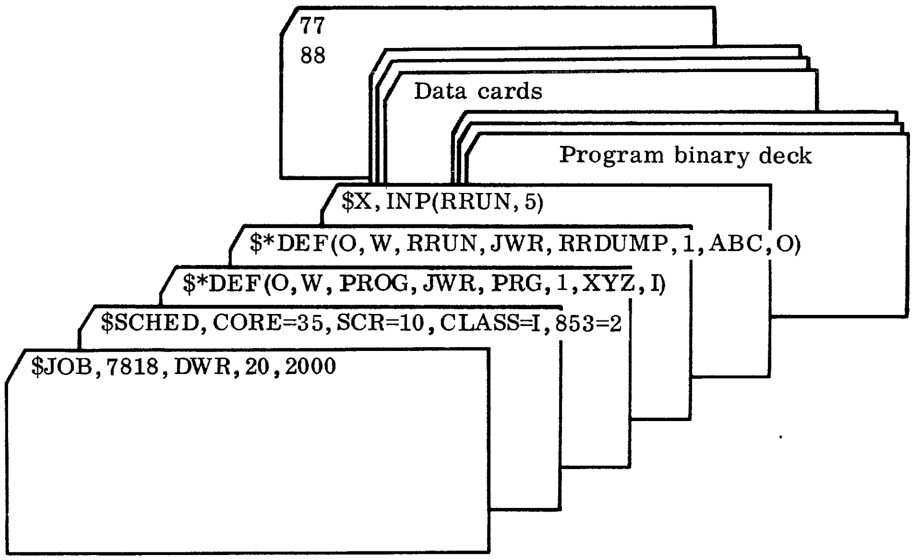
7.6.3 COMPILE AND EXECUTE

COMPILE and EXECUTE With LIST and Disk Object Time CLASS A Storage



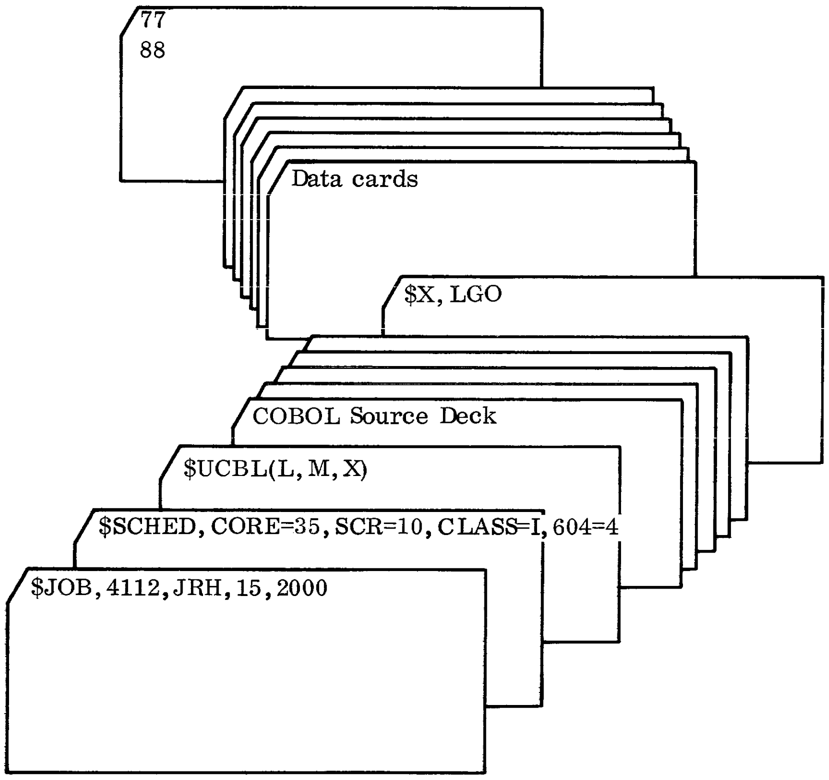
7.6.4
RESTART

RESTART Rerun DUMPFILE on Disk, RESTART at Dump 5

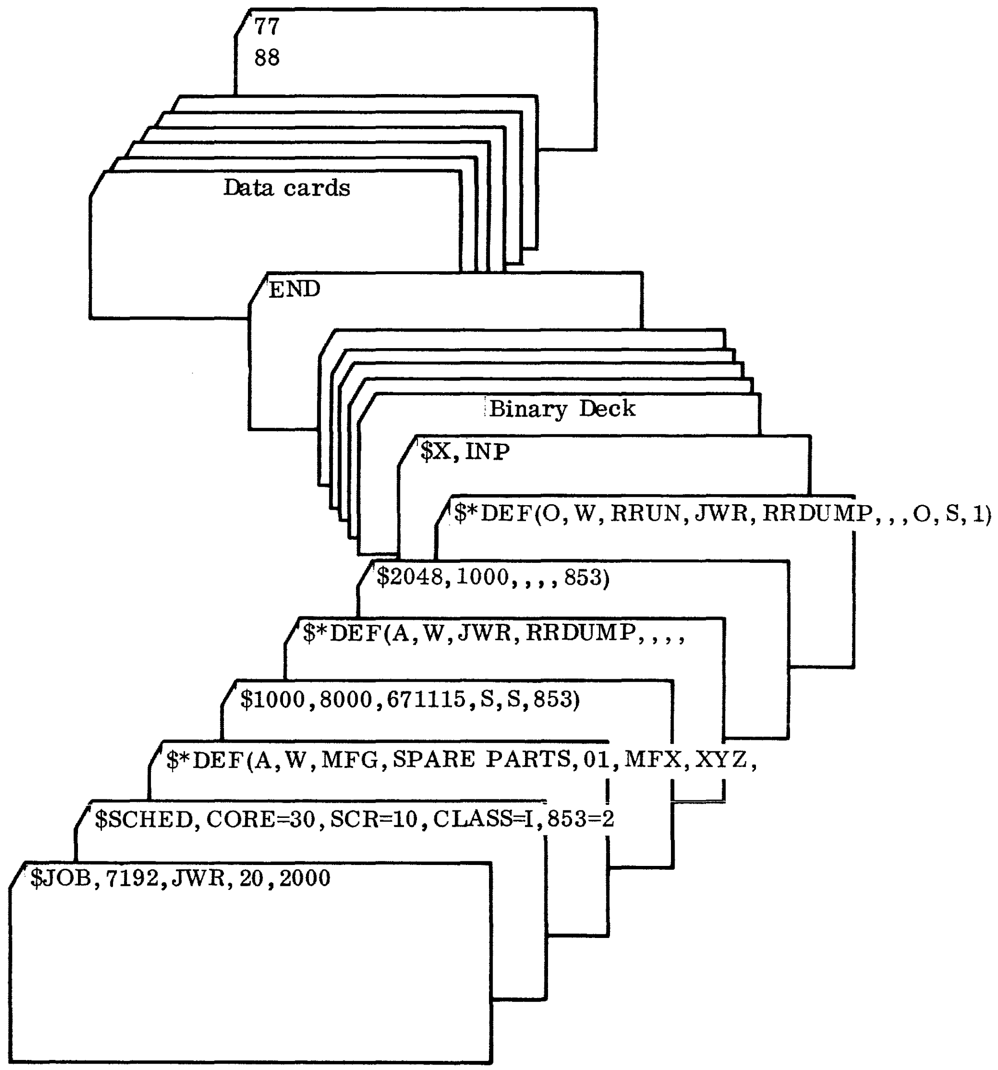

77
88
Data cards
Program binary deck
$X, INP(RRUN, 5)
$*DEF(O, W, RRUN, JWR, RRDUMP, 1, ABC, O)
$*DEF(O, W, PROG, JWR, PRG, 1, XYZ, I)
$SCHED, CORE=35, SCR=10, CLASS=I, 853=2
$JOB, 7818, DWR, 20, 2000

7.6.5
COMPILE AND EXECUTE

COMPILE AND EXECUTE
With LIST, Memory Map, and Tape Object-Time Storage


77
88
Data cards
$X, LGO
COBOL Source Deck
$UCBL(L, M, X)
$SCHED, CORE=35, SCR=10, CLASS=I, 604=4
$JOB, 4112, JRH, 15, 2000

EXECUTE With Object-Time Mass Storage and RERUN on Mass Storage


77
88
Data cards
END
Binary Deck
$X, INP
$*DEF(O, W, RRUN, JWR, RRDUMP, , , O, S, 1)
$2048, 1000, , , , 853)
$*DEF(A, W, JWR, RRDUMP, , , ,
$1000, 8000, 671115, S, S, 853)
$*DEF(A, W, MFG, SPARE PARTS, 01, MFX, XYZ,
$SCHED, CORE=30, SCR=10, CLASS=I, 853=2
$JOB, 7192, JWR, 20, 2000

APPENDIX SECTION

SAMPLE PROGRAMS A

The sample program (PSR-UPDT) listed on the following pages is a Program Summary Report (PSR) update program. Its principal functions are to:

- Maintain the PSR master file as a random access mass storage file.
- Periodically update the master file.
- Extract information from the file.
- Produce reports showing, by product set, the number and status of current PSRs.

PSR-UPDT

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PSR-UPDT.
AUTHOR. DAS-DDJ.
REMARKS.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. 3300.
OBJECT-COMPUTER. 3300.
SPECIAL-NAMES.
    SYSTEM-OUTPUT IS OUT.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT OUT-FILE ASSIGN TO SYSTEM-OUTPUT.
    SELECT INFILE-CARD ASSIGN TO SYSTEM-INPUT.
    SELECT PRIM-SEC-LIST ASSIGN TO DISK LIST
        ACCESS MODE IS RANDOM
        ACTUAL KEY IS PRIM-KEY.
    SELECT PSR-FILE ASSIGN TO DISK PSR
        ACCESS MODE IS RANDOM
        ACTUAL KEY IS PSR-KEY.
    SELECT NUMBER-FILE ASSIGN TO DISK PNUM
        ACCESS MODE IS RANDOM
        ACTUAL KEY IS NUM-KEY.
    SELECT SORT-FILE ASSIGN TO SCRATCH SCR.
    SELECT SORT-OUT ASSIGN TO TAPE 05.
    SELECT P-S-LIST  ASSIGN TO  DISK  LST
        ACCESS MODE IS  RANDOM
        ACTUAL KEY IS  P-KEY.
    SELECT N-FILE  ASSIGN TO DISK PNM
        ACCESS MODE IS SEQUENTIAL
        ACTUAL KEY IS N-KEY.
DATA DIVISION.
FILE SECTION.
FD  OUT-FILE
    LABEL RECORDS ARE OMITTED
    REPORTS ARE DELETE-REPORT-1, LIST-PSR-NUM-REPORT.
FD  INFILE-CARD
    LABEL RECORD IS OMITTED
    DATA RECORD IS INREC.
01  INREC.
    02  CARD-TYPE      PIC X.
        88  LEGAL-CODE VALUES ≠A≠, ≠C≠, ≠D≠, ≠L≠, ≠S≠, ≠R≠.
    02  FILLER         PIC X(79).
```

```
FD  PRIM-SEC-LIST
    BLOCK CONTAINS 24 RECORDS
    RECORD CONTAINS 82 CHARACTERS
    LABEL RECORD IS STANDARD VALUE OF
        ID IS ≠LIST-FILES≠
        OWNER IS ≠PSR≠
        ACCESS-PRIVACY IS ≠APSR≠
        MODIFICATION-PRIVACY IS ≠MPSR≠
    EDITION-NUMBER IS 01
    DATA RECORDS ARE
        HEADER-RECORD, PRIMARY-RECORD,
           SECONDARY-RECORD, EMPTY-RECORD.
01  HEADER-RECORD.
    02  FORWARD-LINK       PIC 9(12).
    02  BACKWARD-LINK      PIC 9(12).
    02  HEADER-LINK      PIC 9(12).
    02  HEADER-LEV-NAME.
        03  SUPP-LEVEL   PIC X.
        03  HEADER-NAME  PIC X(21).
    02  HEADER-SAK       PIC 9(12).
01  PRIMARY-RECORD.
    02  FILLER           PIC X(24).
    02  PRODUCT-NAME     PIC X(10).
    02  MASTER-SAK       PIC 9(12).
    02  MSOS-SAK         PIC 9(12).
    02  RTS-SAK          PIC 9(12).
    02  FILLER           PIC X(12).
01  SECONDARY-RECORD.
    02 FILLER            PIC X(24).
    02 PSR-NUMBER        PIC 9(5).
    02 PSR-SAK           PIC 9(12).
    02 PROD-NAME         PIC X(10).
    02 FILLER            PIC XX.
    02 OP-SYS-NAME       PIC X(6).
    02 LIST-LIT          PIC X(8).
    02 FILLER            PIC X(15).
01  EMPTY-RECORD.
    02  EMPTY-SAK        PIC 9(12).
    02  FILLER           PIC X(70).
FD  PSR-FILE
    RECORD CONTAINS 512 CHARACTERS
    LABEL RECORD IS STANDARD VALUE OF
        ID IS ≠PSR-FILES≠
        OWNER IS ≠PSR≠
        ACCESS-PRIVACY IS ≠APSR≠
```

```
                       MODIFICATION-PRIVACY IS ≠MPSR≠
                   EDITION-NUMBER IS 02
                   DATA RECORD IS PSR-FILE-HEADER,  PSR-FILE-REC.
D00044         01  PSR-FILE-HEADER.
                   02  NEXT-EMPTY      PIC 9(12).
D00045
D00046             02  FILLER          PIC X(500).
D00047         01  PSR-FILE-REC.
D00048             02  PSR-FILE1-MOVE.
D00049                 03  PSR-NO      PIC 9(5).
D00050                 03  DATE-IN     PIC X(8).
D00051                 03  PROD-NM     PIC X(10).
D00052                 03  VERSION-NO          PIC 9V9.
D00053                 03  PROD-NO     PIC X(4).
D00054                 03  OP-SYS-1    PIC X.
D00055                 03 VER-1                PIC 9V9.
D00056                 03  OP-SYS-2    PIC X.
D00057                 03 VER-2                PIC 9V9.
D00058                 03  OP-SYS-3    PIC X.
D00059                 03 VER-3                PIC 9V9.
D00060                 03  REPORT-ORG    PIC X(14).
D00061                 03  EQUAL-PSR     PIC X(5).
D00062             02  PSR-FILE2-MOVE.
D00063                 03  OUT-DTD       PIC X(8).
D00064                 03  SCHED-IMPL.
D00065                     04 SYS-A            PIC 9V9.
D00066                     04 VER-A            PIC 9V9.
D00067                     04 SYS-B            PIC 9V9.
D00068                     04 VER-B            PIC 9V9.
D00069                     04 SYS-C            PIC 9V9.
D00070                     04 VER-C            PIC 9V9.
D00071                 03  TESTED        PIC  X.
D00072                 03  SUM-NO        PIC  999.
D00073             02  DESCRIPTION.
D00074                 03  DESC          PIC X(73) OCCURS 5 TIMES.
D00075             02  PSR-FILE3-MOVE.
D00076                 03  ABS-DT        PIC 9(4).
D00077                 03  PSR-AK-FORW   PIC 9(12).
D00078                 03  PSR-AK-BACK   PIC 9(12).
D00079                 03 OP-KEY1.
D00080                     04  MASTER-LIST-AK PIC 9(12).
D00081                     04  MSOS-LIST-AK PIC 9(12).
D00082                     04  RTS-LIST-AK PIC 9(12).
D00083                 03  OP-KEY2 REDEFINES OP-KEY1.
D00084                     04  OP-KEY PIC 9(12) OCCURS 3 TIMES.
D00085                 03  FILLER        PIC  XX.
```

```
                FD  NUMBER-FILE
                    BLOCK CONTAINS 42 RECORDS
                    RECORD CONTAINS 12 CHARACTERS
                    LABEL RECORD IS STANDARD  VALUE OF
                        ID IS ≠PSR-LIST≠
                        OWNER IS ≠PSR≠
                        ACCESS-PRIVACY IS ≠APSR≠
                        MODIFICATION-PRIVACY IS ≠MPSR≠
                    EDITION-NUMBER IS 03
                    DATA RECORD IS  NUMBER-REC.
D00087          01  NUMBER-REC.
D00088              02  NUM-BLK      PIC 9(7).
D00089              02  NUM-CHAR     PIC 9(5).
                FD  SORT-OUT
                    LABEL RECORD IS OMITTED
                    DATA RECORDS ARE
                        REC-A, REC-C, REC-D, REC-L, REC-U, REC-R.
D00091          01  REC-A.
D00092              02  CODE-ALPHA       PIC X.
D00093              02  CODE-NUM         PIC 9.
D00094              02  REC-A-MOVE1.
D00095                  03  PSR-NUM PIC X(5).
D00096                  03  FILLER PIC X(52).
D00097              02  FILLER           PIC X(21).
D00098          01  REC-C.
D00099              02  FILLER           PIC X(7).
D00100              02  LOGGED-OUT       PIC X(8).
D00101              02  SCHEDULED-IMPL.
D00102                  03  OP-SYSTEM-1 PIC 99.
D00103                  03  VERS-1      PIC 99.
D00104                  03  OP-SYSTEM-2 PIC 99.
D00105                  03  VERS-2      PIC 99.
D00106                  03  OP-SYSTEM-3 PIC 99.
D00107                  03  VERS-3      PIC 99.
D00108              02  TEST              PIC X.
D00109              02  SUM-NUM           PIC 999.
D00110              02  PSR-EQ            PIC X(45).
D00111              02  FILLER            PIC X(4).
D00112          01  REC-D.
D00113              02  FILLER            PIC XX.
D00114              02  DELETE-FIELD  OCCURS 5  TIMES.
D00115                  03 DELETE-PSR-NO  PIC 9(5).
D00116                  03 DELETE-REASON  PIC X(9).
D00117                  03  DELETE-LINKED PIC X.
D00118              02  FILLER           PIC XXX.
```

```
D00119          01  REC-L.
                    02  FILLER              PIC XX.
D00121              02  PRODUCT-LEVEL       PIC X(10).
D00122              02  OP-SYS-LEVEL        PIC X(6).
D00123              02  SUPPORT-LEVEL       PIC X.
D00124              02  FILLER              PIC X(61).
D00125          01  REC-U.
D00126              02  FILLER              PIC X.
D00127              02  U-FIELD             PIC X(15).
D00128              02  U-BIAS              PIC 9(5).
D00129              02  FILLER              PIC X(59).
D00130          01  REC-R.
D00131              02  FILLER PIC X.
D00132              02  OS-FIELD.
D00133                  03  OS-FIELD1.
D00134                      04  OS-FIELD2 PIC X(4).
D00135                      04  OS-FIELD3 PIC X.
D00136                  03  OS-FIELD4 PIC X.
D00137              02  PROD-FIELD PIC X(13).
D00138              02  WHICH-FIELD PIC X(10).
D00139              02  AGE-FIELD PIC X(5).
D00140              02  QUANTITY-FIELD PIC X(7).
D00141              02  FILLER PIC X(48).
                SD  SORT-FILE
                    RECORD CONTAINS 80 CHARACTERS
                    DATA RECORD IS SORT-FILE-REC.
D00143          01  SORT-FILE-REC.
D00144              02  SORT-CODE-1     PIC X.
D00145              02  SORT-CODE-2     PIC X.
D00146              02  SORT-CODE-3     PIC X(5).
D00147              02  FILLER          PIC X(73).
                FD  N-FILE
                    BLOCK CONTAINS 42 RECORDS
                    RECORD CONTAINS 12 CHARACTERS
                    LABEL RECORDS ARE STANDARD VALUE OF
                        ID IS ≠TEMPORY-PSR-LIST≠
                        OWNER  IS  ≠PSR≠
                        ACCESS-PRIVACY IS  ≠APSR≠
                    MODIFICATION-PRIVACY IS ≠MPSR≠
                    EDITION-NUMBER IS 04
                    DATA RECORD  IS  T-REC-1.
D00149          01  T-REC-1          PIC 9(12).
                FD  P-S-LIST
                    BLOCK CONTAINS 24 RECORDS
                    RECORD CONTAINS 82  CHARACTERS
```

```
                      LABEL RECORDS ARE STANDARD VALUE OF
                          ID IS ≠TEMPORY-LIST-FILE≠
                          OWNER IS ≠PSR≠
                          ACCESS-PRIVACY IS  ≠APSR≠
                          MODIFICATION-PRIVACY IS  ≠MPSR≠
                      EDITION-NUMBER IS 05
                      DATA RECORD IS T-REC-2.
D00151            01  T-REC-2          PIC  X(82).
                  WORKING-STORAGE SECTION.
D00153            77  REL-FILES PIC XX.
D00154            77  CTR           PIC  99.
D00155            77 OP-CTR PICTURE IS 9.
D00156            77 IDENT-CTR PICTURE IS 99.
D00157            77 CHECKED-OUT PICTURE IS X(19).
D00158            77  STARS PIC X(6)  VALUE  ≠ **** ≠.
D00159            77  DIAGNOSTIC-U  PIC X(44)  VALUE
                         ≠ILLEGAL UTILITY FUNCTION≠.
D00160            77  DIAGNOSTIC-C  PIC X(44) VALUE
                         ≠ILLEGAL CARD CODE≠.
D00161            77  DIAGNOSTIC-P  PIC X(44) VALUE
                         ≠NEW PRODUCT NAME -- NOT ALLOWED≠.
D00162            77  DIAGNOSTIC-L1  PIC  X(44)  VALUE
                      ≠LEVEL CHANGE - NON-EXISTENT OPERATING SYSTEM≠.
D00163            77  DIAGNOSTIC-L2  PIC  X(44) VALUE
                         ≠LEVEL CHANGE -- NON-EXISTENT PRODUCT NAME≠.
D00164            77  DIAGNOSTIC-P1  PIC  X(44) VALUE
                         ≠PSR NUMBER -- LESS THAN BIAS≠.
D00165            77  DIAGNOSTIC-P2  PIC  X(44) VALUE
                         ≠PSR NUMBER -- NON-EXISTENT FOR DELETE≠.
D00166            77  DIAGNOSTIC-P3  PIC  X(44) VALUE
                         ≠PSR NUMBER -- EXISTING FOR ADD≠.
D00167            77  DIAGNOSTIC-S  PIC  X(44) VALUE
                         ≠ILLEGAL SEQUENCE NUMBER≠.
D00168            77  DIAGNOSTIC-O  PIC X(44) VALUE
                         ≠NO OPERATING SYSTEM≠.
D00169            77  DIAGNOSTIC-E1  PIC X(44)   VALUE
                         ≠EQUATED PSR NUMBER -- NOT IN FILE, IGNORED≠.
D00170            77  DIAGNOSTIC-P4    PIC X(44)    VALUE
                         ≠PSR NUMBER -- NON-EXISTENT FOR CHANGE≠.
D00171            77  BK1-CHO          PIC    9(12)   VALUE  100000.
D00172            77  HOLD-PSR         PIC    9(5).
D00173            77  PSR              PIC    9(5).
D00174            77  ALPHA-SAVE       PIC    X.
D00175            77  READY-PSR        PIC    9(12).
D00176            77  READY-LIST       PIC    9(12).
```

```
D00177          77  GO-KEY           PIC   9.
D00178          77  SAVE-SAK         PIC   9(12).
D00179          77  DELETE-SUB       PIC   9.
D00180          77  BACK-SAVE        PIC   9(12).
D00181          77  BACK-SAVE        PIC   9(12).
D00182          77  I                PIC   9.
D00183          77  TEM-KEY-1        PIC   9(12).
D00184          77  TEM-KEY-2        PIC   9(12).
D00185          77  TEM-KEY-3        PIC   9(12).
D00186          77  TEM-KEY-4        PIC   9(12).
D00187          77  TEM-KEY-5        PIC   9(12).
D00188          77  TEM-KEY-6        PIC   9(12).
D00189          77  TEM-KEY-7        PIC   9(12).
D00190          77  TEM-KEY-8 PIC 9(7).
D00191          77  REPT-HEADING PIC X(17).
D00192          77  OS-NUMBER PIC 9.
D00193          77  TODAYS-DATE PIC 9(4).
D00194          77  DIFF-DATE PIC 9(4).
D00195          77  DESC-CTR PIC 9.
D00196          77  TEM-HOLD-1       PIC   X(82).
D00197          77  TIME-STORE PIC X(8).
D00198          77  DATE-STORE PIC X(8).
D00199          01  P-KEY PIC 9(12).
D00200          01  N-KEY PIC 9(12).
D00201          01  LIST-AREA-FIELD.
D00202              02  LIST-AREA OCCURS 6 TIMES PICTURE IS 9(6).
D00203          01  FIRST-DATE.
D00204              02  MM               PIC 99.
D00205              02  FILLER           PIC X.
D00206              02  DD               PIC 99.
D00207              02  FILLER           PIC X.
D00208              02  YY               PIC 99.
D00209          01  MMTAB.
D00210              02  FILLER           PIC X(36)  VALUE IS
                    ≠000031059090120151181212243273304334≠.
D00211          01  TAB REDEFINES MMTAB.
D00212              02  MTAB             PIC 999  OCCURS 12 TIMES.
D00213          01  UP-KEY1.
D00214              02  U-BLK      PIC 9(7).
D00215              02  U-CH       PIC 9(5).
D00216          01  UP-KEY REDEFINES UP-KEY1 PIC 9(12).
D00217          01  B-C              PIC 9(7)V999.
D00218          01  B-C1 REDEFINES B-C.
D00219              02  BLK          PIC 9(7).
D00220              02  CHAR         PIC 999.
```

```
D00221          01  PRIM-KEY1.
D00222              02  PRIM-KEY-1  PIC 9(7).
D00223              02  PRIM-KEY-2  PIC 9(5).
D00224          01  PRIM-KEY REDEFINES PRIM-KEY1 PIC 9(12).
D00225          01  PSR-KEY1.
D00226              02  PSR-KEY-1   PIC 9(7).
D00227              02  PSR-KEY-2   PIC 9(5).
D00228          01  PSR-KEY REDEFINES PSR-KEY1 PIC 9(12).
D00229          01  NUM-KEY1.
D00230              02  NUM-KEY-1   PIC 9(7).
D00231              02  NUM-KEY-2   PIC 9(5).
D00232          01  NUM-KEY REDEFINES NUM-KEY1 PIC 9(12).
D00233          01  HEADER-RECORD-STORAGE.
D00234              02  EMPTY-HEAD-REC   PIC  9(12).
D00235              02  PLH-SAK          PIC  9(12).
D00236              02  PRIM-EXPAND      PIC  9(12).
D00237              02  H-R-S            PIC  X(34).
D00238              02  H-R-S-SAK        PIC  9(12).
D00239          01  PSR-FILE-HEADER-STORAGE.
D00240              02  EMPTY-PSR-FILE   PIC  9(12).
D00241              02  FIRST-PSR-ENT    PIC  9(12).
D00242              02  PSR-EXPAND       PIC  9(12).
D00243              02  P-F-H-S-H        PIC  X(476).
D00244          01  NUMBER-FILE-STORAGE.
D00245              02  NUM-EXPAND       PIC  9(7).
D00246              02  BIAS             PIC  9(5).
D00247          01  PSR-EQT.
D00248              02  EQUAL-PSRS  PIC  9(5)  OCCURS 9 TIMES
                             INDEXED BY  PSR-INDEX.
D00250          01  HEADER-RECORDT.
D00251              02  FORWARD-LINKT    PIC 9(12).
D00252              02  BACKWARD-LINKT   PIC 9(12).
D00253              02  HEADER-LINKT     PIC 9(12).
D00254              02  SUPP-LEVELT      PIC X.
D00255              02  HEADER-NAMET     PIC X(33)  VALUE  ≠ LIST HEADER≠.
D00256              02  HEADER-SAKT      PIC 9(12).
D00257          01  PRIMARY-RECORDT.
D00258              02  FOR-LINKT        PIC 9(12).
D00259              02  BAK-LINKT        PIC 9(12).
D00260              02  PRODUCT-NAMET    PIC X(10).
D00261              02  MASTER-SAKT      PIC 9(12).
D00262              02  MSOS-SAKT        PIC 9(12).
D00263              02  RTS-SAKT         PIC 9(12).
D00264              02  HEAD-SAKT        PIC X(12).
D00265          01  RPRIMARY-RECORDT REDEFINES PRIMARY-RECORDT.
```

```
        02  FILLER PIC X(34).
        02  R-SAKT PIC 9(12) OCCURS 3 TIMES.
        02  FILLER PIC X(12).
    01  SECONDARY-RECORDT.
        02  FORWARD-LKT      PIC 9(12).
        02  BACKWARD-LKT     PIC 9(12).
        02  PSR-NUMBERT PIC X(5).
        02  PSR-SAKT PIC 9(12).
        02  PROD-NAMET       PIC X(10).
        02  FILLER           PIC XX.
        02  OP-SYS-NAMET     PIC X(6).
        02  FILLER           PIC XX.
        02  LIST-LITERALT    PIC X(9) VALUE  ≠ PSR-LINK≠.
        02  HEADER-ST        PIC 9(12).
    01  PSR-FILE-RECT.
        02  PSR-FILE1-MOVET.
            03  PSR-NOT                 PIC  9(5).
            03  DATE-INT                PIC  X(8).
            03  PROD-NMT                PIC  X(10).
            03  VERSION-NOT             PIC 9V9.
            03  PROD-NOT                PIC  X(4).
            03  OP-SYS-1T               PIC X.
            03 VER-1T                   PIC 9V9.
            03  OP-SYS-2T               PIC  X.
            03 VER-2T                   PIC 9V9.
            03  OP-SYS-3T               PIC  X.
            03 VER-3T                   PIC 9V9.
            03  REPORT-ORGT             PIC  X(14).
            03  EQUAL-PSRT              PIC  9(5).
        02  PSR-FILE2-MOVET.
            03  OUT-DTDT                PIC  X(8).
            03  SCHED-IMPLT.
                04 SYS-AT               PIC 9V9.
                04 VER-AT               PIC 9V9.
                04 SYS-BT               PIC 9V9.
                04 VER-BT               PIC 9V9.
                04 SYS-CT               PIC 9V9.
                04 VER-CT               PIC 9V9.
            03  TESTED-T                PIC  X.
            03  SUM-NOT                 PIC  999.
        02  DESCRIPTIONT.
            03  DESCT                   PIC  X(73) OCCURS 5 TIMES.
        02  PSR-FIL3-MOVET.
            03  AB-DTT                  PIC  9(4).
            03  PSR-AK-FORWT            PIC  9(12).
```

```
D00311              03  PSR-AK-BACKT        PIC  9(12).
D00312              03  MASTER-LIST-AKT     PIC  9(12).
D00313              03  MSOS-LIST-AKT       PIC  9(12).
D00314              03  RTS-LIST-AKT        PIC  9(12).
D00315              03  FILLER              PIC  XX.
D00316      01 SYS-OP-1.
D00317          02 SYS-OP PICTURE IS X(6) OCCURS 3 TIMES.
D00318      01 NO-VER-1.
D00319          02  NO-VER PICTURE IS 9V9 OCCURS 3 TIMES.
D00320      01 IDENT-PSR-STOR.
D00321          02 STOR-IDENT OCCURS 10 TIMES INDEXED BY INDX-STOR.
D00323              03 STOR-IDENT-1 PIC X.
D00324              03  FILLER PIC X.
D00325              03 STOR-IDENT-2 PIC 9(5).
D00326      01 PROD-VER-1.
D00327          02  PROD-VER PICTURE IS 9V9 OCCURS 3 TIMES.
            REPORT SECTION.
            RD  DELETE-REPORT-1
                    PAGE LIMIT IS 60 LINES
                    HEADING 10
                    FIRST DETAIL 12.
            01  LINE NUMBER IS 10
                    TYPE IS PAGE HEADING
                    COLUMN NUMBER IS 37
                    PICTURE IS X(17) SOURCE IS REPT-HEADING.
            01  REPORTED-DELETE
                    TYPE IS DETAIL.
                02  LINE NUMBER IS 12
                    COLUMN NUMBER IS 73
                    PICTURE IS X(8)
                    SOURCE IS SYSTEM-DATE.
                02  LINE NUMBER IS 15
                    COLUMN NUMBER IS 10
                    PICTURE IS X(10) VALUE IS ≠PSR NUMBER≠.
                02  COLUMN NUMBER IS 22
                    PICTURE IS 9(5)
                    SOURCE IS PSR-NOT.
                02  LINE NUMBER IS 17
                    COLUMN NUMBER IS 10
                    PICTURE IS X(7) VALUE IS ≠PRODUCT≠.
                02  COLUMN NUMBER IS 22
                    PICTURE IS X(10)
                    SOURCE IS PROD-NMT.
                02  COLUMN NUMBER IS 35
                    PICTURE IS X(14) VALUE IS ≠VERSION NUMBER≠.
```

```
    02  COLUMN NUMBER IS 51
        PICTURE IS 9.9
        SOURCE IS VERSION-NOT.
    02  COLUMN NUMBER IS 57
        PICTURE IS X(14) VALUE IS ≠PRODUCT NUMBER≠.
    02  COLUMN NUMBER IS 73
        PICTURE IS X(4)
        SOURCE IS PROD-NOT.
01  OP-SYS-RPT
        TYPE IS DETAIL.
    02  LINE NUMBER IS PLUS 2
        COLUMN NUMBER 35
        PICTURE IS X(16) VALUE IS ≠OPERATING SYSTEM≠.
    02  COLUMN NUMBER IS 53
        PICTURE IS X(6)
        SOURCE IS SYS-OP (OP-CTR).
    02  COLUMN NUMBER IS 61
        PICTURE IS X(14) VALUE IS ≠VERSION NUMBER≠.
    02  COLUMN NUMBER IS 77
        PICTURE IS 9.9
        SOURCE IS NO-VER (OP-CTR).
01  RPT-ORG
        TYPE IS DETAIL.
    02  LINE NUMBER IS PLUS 2
        COLUMN NUMBER IS 35
        PICTURE IS X(22) VALUE IS ≠REPORTING ORGANIZATION≠.
    02  COLUMN NUMBER IS 60
        PICTURE IS X(14)
        SOURCE IS REPORT-ORGT.
    02  LINE NUMBER IS PLUS 2
        COLUMN NUMBER IS 35
    PIC X(13) VALUE IS ≠REPORTED DATE≠.
    02  COLUMN NUMBER IS 60
        PICTURE IS X(8)
        SOURCE IS DATE-INT.
    02  LINE NUMBER IS PLUS 2
        COLUMN NUMBER IS 35
        PICTURE IS X(11) VALUE IS ≠PSR SUMMARY≠.
    02  COLUMN NUMBER IS 60
        PICTURE IS 999
        SOURCE IS SUM-NOT.
01  NOT-ANS
        TYPE IS DETAIL.
    02 LINE NUMBER IS PLUS 2
        COLUMN NUMBER IS 10
        PICTURE IS X(12) VALUE IS ≠NOT ANSWERED≠.
01  ANSWERED
        TYPE IS DETAIL.
    02  LINE NUMBER IS PLUS 2
        COLUMN NUMBER IS 10
        PICTURE IS X(20) VALUE IS ≠ANSWERED INFORMATION≠.
    02  LINE NUMBER IS PLUS 2
        COLUMN NUMBER IS 10
        PICTURE IS X(13) VALUE IS ≠DATE ANSWERED≠.
    02  COLUMN NUMBER IS 25
        PICTURE IS X(8)
        SOURCE IS OUT-DTDT.
    02  COLUMN NUMBER IS 36
        PICTURE IS X(19)
        SOURCE IS CHECKED-OUT.
01  OPER-SYS-RPT
        TYPE IS DETAIL.
    02  LINE NUMBER IS PLUS 2
        COLUMN NUMBER IS 30
        PICTURE IS X(21) VALUE IS ≠IMPL OPERATING SYSTEM≠.
    02  COLUMN NUMBER IS 53
        PICTURE IS X(6)
        SOURCE IS SYS-OP (OP-CTR).
    02  COLUMN NUMBER IS 61
        PICTURE IS X(14) VALUE IS ≠VERSION NUMBER≠.
    02  COLUMN NUMBER IS 77
        PICTURE IS 9.9
        SOURCE IS NO-VER (OP-CTR).
    02  LINE NUMBER IS PLUS 1
        COLUMN NUMBER IS 30
        PICTURE IS X(12) VALUE IS ≠IMPL PRODUCT≠.
    02  COLUMN NUMBER IS 44
        PICTURE IS X(10)
        SOURCE IS PROD-NMT.
    02  COLUMN NUMBER IS 61
        PICTURE IS X(14) VALUE IS ≠VERSION NUMBER≠.
    02  COLUMN NUMBER IS 77
        PICTURE IS 9.9
        SOURCE IS PROD-VER (OP-CTR).
01  REL-PSRS
        TYPE IS DETAIL.
    02  LINE NUMBER IS PLUS 3 COLUMN NUMBER IS 10
        PICTURE IS X(12) VALUE IS ≠RELATED PSRS≠.
    02  LINE NUMBER IS PLUS 2 COLUMN NUMBER IS 10
        PICTURE IS X(70)
```

```
                SOURCE IS IDENT-PSR-STOR.
            02  LINE NUMBER IS PLUS 3
                COLUMN NUMBER IS 10
                PICTURE IS X(11) VALUE IS ≠DESCRIPTION≠.
        01  DESCRIP-RPT
                TYPE IS DETAIL.
            02  LINE NUMBER IS PLUS 2
                COLUMN NUMBER IS 10
                PICTURE IS X(73)
                SOURCE IS DESCT (DESC-CTR).
        01  DEL-REASON
                TYPE IS DETAIL.
            02 LINE NUMBER IS PLUS 3
                COLUMN NUMBER IS 10
                PICTURE IS X(17) VALUE IS ≠REASON FOR DELETE≠.
            02  LINE NUMBER IS PLUS 2
                COLUMN NUMBER IS 10
                PICTURE IS X(9)
                SOURCE IS DELETE-REASON (DELETE-SUB).
        RD  LIST-PSR-NUM-REPORT
                PAGE LIMIT IS 60 LINES
                HEADING 10
                FIRST DETAIL 18.
        01  LINE NUMBER IS 10
            TYPE IS PAGE HEADING
            NEXT GROUP IS PLUS 3.
            02  COLUMN NUMBER IS 56
                PICTURE IS X(6)
                SOURCE IS OP-SYS-NAMET.
            02  COLUMN NUMBER IS 63
                PICTURE IS X(13)
                SOURCE IS PRODUCT-NAMET.
            02  COLUMN NUMBER IS 77
                PICTURE IS X(4) VALUE IS ≠PSRS≠.
            02  LINE NUMBER IS 13
                COLUMN IS 53
                PICTURE IS X(13)
                SOURCE IS WHICH-FIELD.
            02  COLUMN NUMBER IS 67
                PICTURE IS X(5)
                SOURCE IS AGE-FIELD.
            02  COLUMN NUMBER IS 73
                PICTURE IS X(8)
                SOURCE IS SYSTEM-DATE.
        01  LISTING-REPORT
        TYPE IS DETAIL.
    02  LINE NUMBER IS PLUS 2
        COLUMN NUMBER IS 10
        PICTURE IS 9(6) SOURCE IS LIST-AREA (1).
    02  COLUMN NUMBER IS 32
        PICTURE IS 9(6) SOURCE IS LIST-AREA (2).
    02  COLUMN NUMBER IS 54
        PICTURE IS 9(6) SOURCE IS LIST-AREA (3).
    02  COLUMN NUMBER IS 76
        PICTURE IS 9(6) SOURCE IS LIST-AREA (4).
    02  COLUMN NUMBER IS 98
        PICTURE IS 9(6) SOURCE IS LIST-AREA (5).
    02  COLUMN NUMBER IS 120
        PICTURE IS 9(6) SOURCE IS LIST-AREA (6).
PROCEDURE DIVISION.
DECLARATIVES.
PRIM-SEC SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON PRIM-SEC-LIST.
PARA-1.
    DISPLAY ≠ ERROR OCCURRED ON I/O OF PRIM-SEC-LIST≠ UPON OUT.
PSR SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON PSR-FILE.
PARA-2.
    DISPLAY ≠ ERROR OCCURRED ON I/O OF PSR-FILE≠ UPON OUT.
NUMBR SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON NUMBER-FILE.
PARA-3.
    DISPLAY ≠ ERROR OCCURRED ON I/O OF NUMBER-FILE≠ UPON OUT.
SORTT SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON SORT-OUT.
PARA-4.
    DISPLAY ≠ ERROR OCCURRED ON I/O OF SORT-OUT≠ UPON OUT.
END DECLARATIVES.
READ-CARD SECTION.
SORT-INPUT.
    MOVE SYSTEM-DATE TO FIRST-DATE.
    COMPUTE TODAYS-DATE = ((YY - 66) * 365) + MTAB (MM) + DD.
    OPEN INPUT INFILE-CARD.
    SORT SORT-FILE ON ASCENDING KEY SORT-CODE-1 SORT-CODE-2
        SORT-CODE-3 INPUT PROCEDURE IS INPUT-PROCEDURE
        GIVING SORT-OUT.
    CLOSE INFILE-CARD.
READ-TAPE SECTION.
TAPE-OPEN.
    OPEN INPUT SORT-OUT.
```

```
           READ SORT-OUT AT END GO TO LAST-CARD.
           IF CODE-ALPHA = ≠$≠ GO TO UTILITY.
           MOVE CODE-ALPHA TO ALPHA-SAVE.
       FILE-OPEN.
           OPEN I-O PRIM-SEC-LIST, PSR-FILE, NUMBER-FILE.
           MOVE BK1-CHO TO PRIM-KEY, NUM-KEY, PSR-KEY.
           READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
           READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
           READ NUMBER-FILE INVALID KEY PERFORM PROGRAM-BUG.
           MOVE HEADER-RECORD TO HEADER-RECORD-STORAGE.
           MOVE PSR-FILE-HEADER TO PSR-FILE-HEADER-STORAGE.
           MOVE NUMBER-REC TO NUMBER-FILE-STORAGE.
           GO TO TAPE-CHECK.
       TAPE-INPUT.
           PERFORM READ-ROUTINE.
       TAPE-CHECK.
           IF CODE-ALPHA = ≠A≠ GO TO ADD-PROCESSOR.
           IF CODE-ALPHA = ≠C≠ GO TO CHANGE-PROCESSOR.
           IF CODE-ALPHA = ≠D≠ GO TO DELETE-PROCESSOR.
           IF CODE-ALPHA = ≠L≠ GO TO LEVEL-PROCESSOR.
           IF CODE-ALPHA = ≠R≠ GO TO REPORT-PROCESSOR.
           PERFORM PROGRAM-BUG.
       LAST-CARD.
           MOVE BK1-CHO TO PRIM-KEY, PSR-KEY, NUM-KEY.
           WRITE NUMBER-REC FROM NUMBER-FILE-STORAGE INVALID KEY
               PERFORM PROGRAM-BUG.
           WRITE PSR-FILE-REC FROM PSR-FILE-HEADER-STORAGE INVALID KEY
               PERFORM PROGRAM-BUG.
           WRITE HEADER-RECORD FROM HEADER-RECORD INVALID KEY
               PERFORM PROGRAM-BUG.
           CLOSE SORT-OUT, PRIM-SEC-LIST, PSR-FILE, NUMBER-FILE.
       ALTER-STOP.
           GO TO STOP-PAR.
       STOP-PAR.
           STOP RUN.
       ADDING SECTION.
       ADD-PROCESSOR.
           MOVE 1 TO GO-KEY.
           IF CODE-NUM = 0 MOVE REC-A-MOVE1 TO PSR-FILE1-MOVE,
               PSR-FILE1-MOVET PERFORM READ-ROUTINE ELSE DISPLAY STARS,
               DIAGNOSTIC-S, STARS, REC-A UPON OUT GO TO CARD-SEQ-ERROR.
       IF-CHECKER.
           IF PSR-NUM NOT EQUAL TO PSR-NO GO TO ADD-END.
           IF CODE-NUM = 1 OR 2 OR 3 OR 4 OR 5 GO TO ADD-PROC.
           DISPLAY STARS, DIAGNOSTIC-S, STARS, REC-A UPON OUT.
```

```
    GO TO TAPE-INPUT.
ADD-PROC.
    MOVE REC-A-MOVE1 TO DESC (CODE-NUM), DESCT (CODE-NUM).
    PERFORM READ-ROUTINE.
    GO TO IF-CHECKER.
ADD-END.
    MOVE DATE-INT  TO FIRST-DATE.
    COMPUTE AB-DTT = ((YY - 66) * 365) + MTAB (MM) + DD.
    MOVE PSR-NOT TO PSR.
    COMPUTE B-C = (((PSR - BIAS) * 12) + 504) / 504.
    IF B-C IS NEGATIVE DISPLAY STARS, DIAGNOSTIC-P1, STARS, ≠AO≠,
        PSR-FILE1-MOVET UPON OUT GO TO TAPE-CHECK.
    MOVE BLK TO NUM-KEY-1.
    MOVE CHAR TO NUM-KEY-2.
ADD-END-1.
    READ NUMBER-FILE INVALID KEY PERFORM EXPAND-NUM-FILE
    GO TO ADD-END-1.
    IF NUMBER-REC IS NOT EQUAL TO ZEROES DISPLAY STARS,
        DIAGNOSTIC-P3, STARS, ≠AO≠, PSR-FILE1-MOVET UPON OUT
        GO TO TAPE-INPUT.
ADD-END-2.
    MOVE PSR-NOT TO PSR-NUMBERT.
    MOVE PLH-SAK TO PRIM-KEY, SAVE-SAK.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    IF FORWARD-LINK = BACKWARD-LINK AND HEADER-SAK
        GO TO  BUILD-PRIM-ENTRY.
    MOVE FORWARD-LINK TO PRIM-KEY.
CHECK-ANOTHER.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    IF PROD-NMT = PRODUCT-NAME GO TO FOUND-PROD.
    MOVE FORWARD-LINK TO PRIM-KEY.
    IF PRIM-KEY IS NOT EQUAL TO SAVE-SAK GO TO CHECK-ANOTHER.
BUILD-PRIM-ENTRY.
    MOVE SPACES TO PRODUCT-NAMET.
    DISPLAY ≠ NEW PRODUCT NAME≠ PROD-NMT ≠ TO BE ENTERED ∧≠
        ≠RESPOND OK, NO, OR CORRECT SPELLING≠.
    ACCEPT PRODUCT-NAMET.
    IF PRODUCT-NAMET = ≠OK≠ GO TO BLD-ENTRY.
    IF PRODUCT-NAMET = ≠NO≠ DISPLAY STARS, DIAGNOSTIC-P, STARS,
        ≠AO≠ PSR-FILE1-MOVET UPON OUT GO TO TAPE-CHECK.
    MOVE PRODUCT-NAMET TO PROD-NMT.
    GO TO ADD-END-2.
BLD-ENTRY.
    PERFORM GET-EMPTY-LIST. MOVE ALL ZEROES TO PRIMARY-RECORDT.
    MOVE PRIM-KEY TO BAK-LINKT.
```

```
           MOVE HEADER-SAK TO HEAD-SAKT, FOR-LINKT.
           MOVE PROD-NMT TO PRODUCT-NAMET.
           MOVE READY-LIST TO FORWARD-LINK.
           WRITE PRIMARY-RECORD INVALID KEY PERFORM PROGRAM-BUG.
           MOVE FOR-LINKT TO PRIM-KEY.
           READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
           MOVE READY-LIST TO BACKWARD-LINK.
           WRITE PRIMARY-RECORD INVALID KEY PERFORM PROGRAM-BUG.
           MOVE READY-LIST TO PRIM-KEY.
           MOVE PRIMARY-RECORDT TO PRIMARY-RECORD.
           WRITE PRIMARY-RECORD INVALID KEY PERFORM PROGRAM-BUG.
       FOUND-PROD.
           PERFORM GET-EMPTY-PSR. MOVE PSR-NOT    TO PSR-NUMBERT.
           MOVE READY-PSR TO NUMBER-REC, PSR-SAKT.
           WRITE NUMBER-REC INVALID KEY PERFORM PROGRAM-BUG.
           MOVE PRIM-KEY TO TEM-KEY-1.
           MOVE PROD-NMT TO PROD-NAMET.
           IF OP-SYSTEM-1 = SPACES NEXT SENTENCE ELSE MOVE MASTER-SAK TO
               PRIM-KEY MOVE ≠MASTER≠ TO OP-SYS-NAMET PERFORM SYS-FOUND.
           IF OP-SYSTEM-2 = SPACES NEXT SENTENCE ELSE MOVE MSOS-SAK TO
               PRIM-KEY  MOVE ≠MSOS≠ TO OP-SYS-NAMET PERFORM SYS-FOUND.
           IF OP-SYSTEM-3 = SPACES NEXT SENTENCE ELSE MOVE RTS-SAK TO
               PRIM-KEY MOVE ≠RTS≠ TO OP-SYS-NAMET PERFORM SYS-FOUND.
           IF SPACES = OP-SYSTEM-1 OP-SYSTEM-2 AND OP-SYSTEM-3 DISPLAY
               STARS, DIAGNOSTIC-C, STARS, REC-A UPON OUT.
           IF EQUAL-PSRT = SPACES NEXT SENTENCE ELSE PERFORM
               EQ-PSR-NUM.
           MOVE READY-PSR      TO PSR-KEY.
           WRITE PSR-FILE-REC FROM PSR-FILE-RECT INVALID KEY PERFORM
           PROGRAM-BUG. MOVE TEM-KEY-1 TO PRIM-KEY.
           GO TO TAPE-CHECK.
       EQ-PSR-NUM.
           MOVE EQUAL-PSRT TO PSR.
           COMPUTE B-C = (((PSR - BIAS) * 12) + 504) / 504.
           IF B-C IS NEGATIVE DISPLAY STARS, DIAGNOSTIC-E1, STARS, ≠AO≠,
               PSR-FILE1-MOVET UPON OUT GO TO EQ-PSR-END.
           MOVE BLK TO NUM-KEY-1.
           MOVE CHAR TO NUM-KEY-2.
           READ NUMBER-FILE INVALID KEY MOVE ZERO TO NUMBER-REC.
           IF NUMBER-REC = ZERO DISPLAY STARS, DIAGNOSTIC-E1, STARS,
               ≠AO≠, PSR-FILE1-MOVET UPON OUT GO TO EQ-PSR-END
           MOVE NUMBER-REC TO PSR-AK-BACKT, PSR-KEY.
           READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
           IF PSR-AK-FORW = ZERO MOVE PSR-KEY TO PSR-AK-BACK,
               PSR-AK-FORW.
```

```
    MOVE PSR-AK-FORW TO SAVE-SAK, PSR-AK-FORWT.
    MOVE READY-PSR TO PSR-AK-FORW.
    WRITE PSR-FILE-REC INVALID KEY PERFORM PROGRAM-BUG.
    MOVE SAVE-SAK TO PSR-KEY.
    READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
    MOVE READY-PSR TO PSR-AK-BACK.
    WRITE PSR-FILE-REC INVALID KEY PERFORM PROGRAM-BUG.
EQ-PSR-END.
    EXIT.
CHANGE-PROC SECTION.
CHANGE-PROCESSOR.
    MOVE PSR-NUM TO PSR.
    MOVE 2 TO GO-KEY.
    COMPUTE B-C = (((PSR - BIAS) * 12) + 504) / 504.
    IF CODE-NUM NOT EQUAL TO ZERO DISPLAY STARS, DIAGNOSTIC-S,
        STARS, REC-A UPON OUT GO TO CARD-SEQ-ERROR.
    IF B-C IS NEGATIVE DISPLAY STARS, DIAGNOSTIC-P1, STARS,
        REC-A UPON OUT GO TO CARD-SEQ-ERROR.
    MOVE BLK  TO NUM-KEY-1.
    MOVE CHAR TO NUM-KEY-2.
    READ NUMBER-FILE INVALID KEY MOVE ZERO TO NUMBER-REC.
    IF NUMBER-REC = ZERO DISPLAY STARS, DIAGNOSTIC-P4, STARS,
        REC-A UPON OUT GO TO CARD-SEQ-ERROR
    MOVE NUMBER-REC TO PSR-KEY. MOVE PSR-EQ TO PSR-EQT.
    READ PSR-FILE INTO PSR-FILE-RECT INVALID KEY PERFORM
    PROGRAM-BUG.
    IF LOGGED-OUT = SPACES NEXT SENTENCE ELSE MOVE LOGGED-OUT
        TO OUT-DTOT.
    IF OP-SYSTEM-1 = SPACES NEXT SENTENCE ELSE MOVE OP-SYSTEM-1
        TO SYS-AT.
    IF VERS-1 = SPACES NEXT SENTENCE ELSE MOVE VERS-1 TO VER-AT.
    IF OP-SYSTEM-2 = SPACES NEXT SENTENCE ELSE MOVE OP-SYSTEM-2
        TO SYS-BT.
    IF VERS-2 = SPACES NEXT SENTENCE ELSE MOVE VERS-2 TO VER-BT.
    IF OP-SYSTEM-3 = SPACES NEXT SENTENCE ELSE MOVE OP-SYSTEM-3
        TO SYS-CT.
    IF VER-3 = SPACES NEXT SENTENCE ELSE MOVE VER-3 TO VER-CT.
    IF TEST = SPACES NEXT SENTENCE ELSE MOVE TEST TO TESTED-T.
    IF SUM-NUM = SPACES NEXT SENTENCE ELSE MOVE SUM-NUM TO
        SUM-NOT. MOVE PSR-KEY TO TEM-KEY-3.
CHANGE-CHECKER.
    PERFORM READ-ROUTINE.
    IF PSR-NUM NOT EQUAL TO PSR-NOT GO TO CHANGE-END.
    IF CODE-NUM = 1 2 3 4 OR 5 MOVE REC-A-MOVE1 TO DESCT
        (CODE-NUM) GO TO CHANGE-CHECKER.
    DISPLAY STARS, DIAGNOSTIC-S, STARS, REC-C UPON OUT.
```

```
           GO TO TAPE-INPUT.
       CHANGE-END.
           IF OP-SYSTEM-1 = ZEROES GO TO SECOND-CHECK.
           IF MASTER-LIST-AK = ZEROES MOVE ≠MASTER≠ TO OP-SYS-NAMET
           PERFORM NEED-SECONDARY-LIST.
       SECOND-CHECK.
           IF OP-SYSTEM-2 = ZEROES GO TO THIRD-CHECK.
           IF MSOS-LIST-AK = ZEROES MOVE ≠MSOS≠ TO OP-SYS-NAMET
           PERFORM NEED-SECONDARY-LIST.
       THIRD-CHECK.
           IF OP-SYSTEM-3 = ZEROES GO TO PSR-LINKAGE.
           IF RTS-LIST-AK = ZEROES MOVE ≠RTS≠ TO OP-SYS-NAMET.
           PERFORM NEED-SECONDARY-LIST.
       PSR-LINKAGE. MOVE PSR-KEY TO TEM-KEY-3.
           IF PSR-AK-FORWT = ZEROES AND PSR-AK-BACKT GO TO
               NO-PSR-LINK.
           IF PSR-AK-FORWT IS NOT EQUAL TO PSR-AK-BACKT GO TO BACK-CK.
           MOVE PSR-AK-FORWT TO PSR-KEY.
           READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
           PERFORM MOVE-DATA.
           WRITE PSR-FILE-REC INVALID KEY PERFORM PROGRAM-BUG.
           IF EQUAL-PSRS (1) = SPACES GO TO NO-OTHER-PSR.
           SEARCH EQUAL-PSRS AT END GO TO NO-PSR-LINK WHEN EQUAL-PSRS
               (PSR-INDEX)
               = PSR-NO MOVE ZEROES TO EQUAL-PSRS (PSR-INDEX) GO TO
               NO-PSR-LINK.
       BACK-CK.
           MOVE PSR-AK-BACKT TO PSR-KEY.
           READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
           PERFORM MOVE-DATA.
           WRITE PSR-FILE-REC INVALID KEY PERFORM PROGRAM-BUG.
           IF PSR-AK-BACK = TEM-KEY-3 GO TO   BACK-CK-1.
           IF EQUAL-PSRS (1) = SPACES GO TO BACK-CK.
           SEARCH EQUAL-PSRS AT END GO TO BACK-CK WHEN EQUAL-PSRS
               (PSR-INDEX)
               = PSR-NO MOVE ZEROES TO EQUAL-PSRS (PSR-INDEX) GO TO
               BACK-CK.
       BACK-CK-1.
           IF EQUAL-PSRS (1) = SPACES GO TO NO-OTHER-PSR.
           SEARCH EQUAL-PSRS AT END GO TO NO-PSR-LINK WHEN EQUAL-PSRS
               (PSR-INDEX)
               = PSR-NO MOVE ZEROES TO EQUAL-PSRS (PSR-INDEX).
       NO-PSR-LINK.
           MOVE 1 TO DELETE-SUB.
       NO-PSR-LINK-1.
```

```
    IF DELETE-SUB GREATER THAN 9 GO TO NO-OTHER-PSR.
    MOVE EQUAL-PSRS (DELETE-SUB) TO PSR.
    COMPUTE B-C = (((PSR - BIAS) * 12) + 504) / 504.
    IF B-C IS NEGATIVE DISPLAY STARS, DIAGNOSTIC-E1, STARS,
        PROD-NOT, EQUAL-PSRS (DELETE-SUB) UPON OUT ADD 1 TO
        DELETE-SUB GO TO NO-PSR-LINK-1.
    MOVE BLK TO NUM-KEY-1.
    MOVE CHAR TO NUM-KEY-2.
    READ NUMBER-FILE INVALID KEY MOVE ZEROES TO NUMBER-REC.
    IF NUMBER-REC = ZERO DISPLAY STARS, DIAGNOSTIC-P4, STARS,
        REC-A UPON OUT ADD 1 TO DELETE-SUB GO TO NO-PSR-LINK-1.
    MOVE NUMBER-REC TO PSR-KEY.
    READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
MOVE-DATA.
    IF LOGGED-OUT = SPACES NEXT SENTENCE ELSE MOVE LOGGED-OUT
        TO OUT-DTD.
    IF OP-SYSTEM-1 = SPACES NEXT SENTENCE ELSE MOVE OP-SYSTEM-1
        TO SYS-A.
    IF VERS-1 = SPACES NEXT SENTENCE ELSE MOVE VERS-1 TO VER-A.
    IF OP-SYSTEM-2 = SPACES NEXT SENTENCE ELSE MOVE OP-SYSTEM-2
        TO SYS-B.
    IF  VERS-2 = SPACES NEXT SENTENCE ELSE MOVE VERS-2 TO VER-B.
    IF OP-SYSTEM-3 = SPACES NEXT SENTENCE ELSE MOVE OP-SYSTEM-3
        TO SYS-C.
    IF VER-3 = SPACES NEXT SENTENCE ELSE MOVE VER-3 TO VER-C.
    IF TEST = SPACES NEXT SENTENCE ELSE MOVE TEST TO TESTED.
    IF SUM-NUM = SPACES NEXT SENTENCE ELSE MOVE SUM-NUM TO
        SUM-NO.
MOVE-DATA-1.
    IF PSR-AK-FORW = ZERO GO TO MOVE-DATA-3.
    MOVE PSR-AK-FORW TO PSR-KEY, TEM-KEY-7.
    READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG
    PERFORM MOVE-DATA.
    IF PSR-AK-FORW = NUMBER-REC GO TO EQUAL-PSR-LINKAGE.
MOVE-DATA-2.
        MOVE PSR-AK-FORW TO PSR-KEY.
        READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
        PERFORM MOVE-DATA. IF PSR-AK-FORW = NUMBER-REC NEXT
        SENTENCE ELSE WRITE PSR-FILE-REC INVALID KEY PERFORM
        PROGRAM-BUG GO TO MOVE-DATA-2.
MOVE-DATA-3.
    MOVE TEM-KEY-3 TO PSR-AK-FORW.
    MOVE PSR-AK-BACKT TO PSR-AK-BACK.
    WRITE PSR-FILE-REC INVALID KEY PERFORM PROGRAM-BUG.
    MOVE PSR-KEY TO PSR-AK-BACKT.
    MOVE PSR-KEY TO TEM-KEY-7.
    MOVE PSR-AK-BACKT TO PSR-KEY.
    READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
    MOVE TEM-KEY-7 TO  PSR-AK-FORW.
    WRITE PSR-FILE-REC INVALID KEY PERFORM PROGRAM-BUG.
    ADD 1 TO DELETE-SUB.
    GO TO NO-PSR-LINK-1.
EQUAL-PSR-LINKAGE.
    MOVE TEM-KEY-3 TO PSR-AK-FORW.
    MOVE PSR-AK-BACKT TO TEM-KEY-6.
    MOVE TEM-KEY-7 TO PSR-AK-BACKT.
    WRITE PSR-FILE-REC INVALID KEY PERFORM PROGRAM-BUG.
    MOVE TEM-KEY-6 TO PSR-KEY.
    READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG
    MOVE NUMBER-REC TO PSR-AK-FORW.
    WRITE PSR-FILE-REC INVALID KEY PERFORM PROGRAM-BUG.
    MOVE NUMBER-REC TO PSR-KEY.
    READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
    MOVE TEM-KEY-6 TO PSR-AK-BACK.
    WRITE PSR-FILE-REC INVALID KEY PERFORM PROGRAM-BUG
    MOVE TEM-KEY-7 TO PSR-KEY.
    WRITE PSR-FILE-REC INVALID KEY PERFORM PROGRAM-BUG.
    ADD 1 TO DELETE-SUB.
    GO TO NO-PSR-LINK-1.
NO-OTHER-PSR.
    MOVE TEM-KEY-1 TO PSR-KEY.
    WRITE PSR-FILE-REC FROM PSR-FILE-RECT INVALID KEY
        PERFORM PROGRAM-BUG.
    GO TO TAPE-CHECK.
NEED-SECONDARY-LIST SECTION.
NEED-LIST-1.
    IF  MASTER-LIST-AKT IS NOT ZERO
        MOVE MASTER-LIST-AKT TO  TEM-KEY-1  GO TO NEED-LIST-2.
    IF  MSOS-LIST-AKT  IS NOT ZERO
        MOVE MSOS-LIST-AKT TO TEM-KEY-1  GO TO NEED-LIST-2.
    IF  RTS-LIST-AKT  IS NOT ZERO
        MOVE RTS-LIST-AKT TO  TEM-KEY-1  GO TO  NEED-LIST-2.
    PERFORM PROGRAM-BUG.
NEED-LIST-2.
    MOVE  PRIM-KEY TO TEM-KEY-2.
    MOVE  TEM-KEY-1  TO PRIM-KEY.
    READ  PRIM-SEC-LIST  INVALID KEY PERFORM PROGRAM-BUG.
    MOVE  HEADER-SAK TO PRIM-KEY.
    READ  PRIM-SEC-LIST INVALID KEY  PERFORM PROGRAM-BUG.
    MOVE HEADER-LINK TO PRIM-KEY, TEM-KEY-1.
```

```
           READ  PRIM-SEC-LIST INVALID KEY PERFORM  PROGRAM-BUG.
           MOVE  PSR-KEY TO  PSR-SAKT.
           IF  OP-SYS-NAMET = ≠MASTER≠ MOVE MASTER-SAK TO PRIM-KEY.
           IF  OP-SYS-NAMET = ≠MSOS≠   MOVE MSOS-SAK TO  PRIM-KEY.
           IF  OP-SYS-NAMET = ≠RTS≠    MOVE RTS-SAK  TO  PRIM-KEY.
           MOVE PROD-NMT TO  PROD-NAMET.
           MOVE PSR-NOT  TO  PSR-NUMBERT.
           PERFORM SYS-FOUND.
           IF  OP-SYS-NAMET  = ≠MASTER≠ MOVE READY-LIST TO MASTER-SAKT.
           IF  OP-SYS-NAMET  = ≠MSOS≠  MOVE READY-LIST TO  MSOS-SAKT.
           IF  OP-SYS-NAMET  = ≠RTS≠ MOVE READY-LIST TO RTS-SAKT.
           MOVE TEM-KEY-2 TO PRIM-KEY.
       SYS-FOUND SECTION.
       SECONDARY-ENTRY-INSERT.
           IF  PRIM-KEY = ZERO PERFORM SECONDARY-HEADER-INSERT
               GO TO  SEC-ENT-IN-1.
           READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
       SEC-ENT-IN-1.
           PERFORM GET-EMPTY-LIST.
           IF  FORWARD-LINK = BACKWARD-LINK AND  HEADER-SAK
               GO TO SEC-ENT-IN-3.
       SEC-ENT-IN-2.
           MOVE BACKWARD-LINK TO PRIM-KEY.
           READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
           IF  PSR-NUMBER LESS THAN PSR-NUMBERT GO TO SEC-ENT-IN-3.
           IF  BACKWARD-LINK NOT EQUAL TO HEADER-SAK
               GO TO SEC-ENT-IN-2.
           MOVE BACKWARD-LINK TO PRIM-KEY.
           READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
       SEC-ENT-IN-3.
           MOVE FORWARD-LINK TO FORWARD-LKT.
           MOVE READY-LIST TO FORWARD-LINK.
           MOVE HEADER-SAK TO HEADER-ST.
           MOVE PRIM-KEY TO BACKWARD-LKT.
           WRITE SECONDARY-RECORD INVALID KEY PERFORM PROGRAM-BUG.
           MOVE FORWARD-LKT TO PRIM-KEY.
           READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
           MOVE READY-LIST TO BACKWARD-LINK.
           WRITE SECONDARY-RECORD INVALID KEY PERFORM PROGRAM-BUG.
           MOVE READY-LIST TO PRIM-KEY.
           WRITE SECONDARY-RECORD FROM SECONDARY-RECORDT
               INVALID KEY PERFORM  PROGRAM-BUG.
           IF  OP-SYS-NAMET = ≠MASTER≠ MOVE READY-LIST TO
               MASTER-LIST-AKT.
           IF  OP-SYS-NAMET = ≠MSOS≠ MOVE READY-LIST TO MSOS-LIST-AKT.
```

```
          IF  OP-SYS-NAMET = ≠RTS≠ MOVE READY-LIST TO RTS-LIST-AKT.
      SECONDARY-HEADER-INSERT SECTION.
      SEC-HEAD-IN.
          PERFORM GET-EMPTY-LIST.
          IF  OP-SYS-NAMET = ≠MASTER≠ MOVE READY-LIST TO MASTER-SAK.
          IF  OP-SYS-NAMET = ≠MSOS≠   MOVE READY-LIST TO MSOS-SAK.
          IF  OP-SYS-NAMET = ≠RTS≠    MOVE READY-LIST TO RTS-SAK.
          MOVE TEM-KEY-1 TO PRIM-KEY, HEADER-LINKT.
          WRITE PRIMARY-RECORD INVALID KEY PERFORM PROGRAM-BUG.
          MOVE READY-LIST TO PRIM-KEY, FORWARD-LINKT,
              BACKWARD-LINKT, HEADER-SAKT.
          MOVE  ≠A≠ TO SUPP-LEVELT.
          MOVE HEADER-RECORDT  TO HEADER-RECORD.
          WRITE HEADER-RECORD INVALID KEY PERFORM PROGRAM-BUG.
      DELETING SECTION.
      DELETE-PROCESSOR.
          MOVE 1 TO DELETE-SUB.
          MOVE ≠PSR DELETE REPORT≠ TO REPT-HEADING.
      DELETE-PROC.
          IF DELETE-LINKED (DELETE-SUB) = SPACES NEXT SENTENCE ELSE
              GO TO DELETE-ALL-PROC.
          MOVE DELETE-PSR-NO (DELETE-SUB) TO PSR.
          COMPUTE B-C = (((PSR - BIAS) * 12) + 504) / 504.
          MOVE BLK TO NUM-BLK.
          MOVE CHAR TO NUM-CHAR.
          MOVE NUMBER-REC TO NUM-KEY.
          READ NUMBER-FILE INVALID KEY PERFORM PROGRAM-BUG.
          MOVE NUMBER-REC TO PSR-KEY, TEM-KEY-1.
          MOVE ZEROES TO NUMBER-REC.
          WRITE NUMBER-REC INVALID KEY PERFORM PROGRAM-BUG.
          READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
          MOVE PSR-FILE-REC TO PSR-FILE-RECT.
          MOVE SPACES TO PSR-FILE-REC.
          MOVE EMPTY-PSR-FILE TO NEXT-EMPTY.
          MOVE TEM-KEY-1 TO READY-PSR.
          WRITE PSR-FILE-HEADER INVALID KEY PERFORM PROGRAM-BUG.
          MOVE PSR-AK-BACKT TO PSR-KEY.
          READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
          MOVE PSR-AK-FORWT TO PSR-AK-FORW.
          WRITE PSR-FILE-REC INVALID KEY PERFORM PROGRAM-BUG.
          MOVE PSR-AK-FORWT TO PSR-KEY.
          READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
          MOVE PSR-AK-BACKT TO PSR-AK-BACK.
          WRITE PSR-FILE-REC INVALID KEY PERFORM PROGRAM-BUG.
          IF MASTER-LIST-AKT IS NOT EQUAL TO ZERO MOVE MASTER-LIST-AKT
```

```
        TO PRIM-KEY PERFORM DELETE-LIST.
    IF MSOS-LIST-AKT IS NOT EQUAL TO ZERO MOVE MSOS-LIST-AKT TO
        PRIM-KEY PERFORM DELETE-LIST.
    IF RTS-LIST-AKT IS NOT EQUAL TO ZERO MOVE RTS-LIST-AKT TO
        PRIM-KEY PERFORM DELETE-LIST.
    PERFORM DELETE-PART-2.
    PERFORM DELETE-REPORT.
    ADD 1 TO DELETE-SUB.
    IF DELETE-SUB IS GREATER THAN 5 GO TO TAPE-INPUT ELSE
        GO TO DELETE-PROC.
DELETE-ALL-PROC. MOVE SPACES TO IDENT-PSR-STOR.
DELETE-ALL-1.
    PERFORM GET-AK.
    PERFORM GET-PSR.
    IF MASTER-LIST-AKT IS NOT EQUAL TO ZERO MOVE MASTER-LIST-AKT
        TO PRIM-KEY PERFORM DELETE-LIST.
    IF MSOS-LIST-AKT IS NOT EQUAL TO ZERO MOVE MSOS-LIST-AKT
        TO PRIM-KEY PERFORM DELETE-LIST.
    IF RTS-LIST-AKT IS NOT EQUAL TO ZERO MOVE RTS-LIST-AKT
        TO PRIM-KEY PERFORM DELETE-LIST.
DELETE-PART-1.
    MOVE PSR-AK-BACKT TO PSR-KEY.
    MOVE 1 TO CTR.
DELETE-PART-2.
    READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
    MOVE PSR-NO TO STOR-IDENT-2 (CTR).
    MOVE TEM-KEY-2 TO TEM-KEY-5.
    WRITE PSR-FILE-REC INVALID KEY PERFORM PROGRAM-BUG.
    IF CTR = 10 GO TO DELETE-PART-3. ADD 1 TO CTR.
    IF PSR-AK-BACK  = TEM-KEY-2 NEXT SENTENCE ELSE
    MOVE ≠,≠ TO STOR-IDENT-1 (CTR)
        MOVE PSR-AK-BACK TO PSR-KEY GO TO DELETE-PART-2.
DELETE-PART-3.
    PERFORM DELETE-RPT.
DELETE-PART-4.
    PERFORM DELETE-ALL-1.
    PERFORM DELETE-PART-1.
    SET INDX-STOR TO 1.
    SEARCH STOR-IDENT AT END GO TO DELETE-PART-5 WHEN
        STOR-IDENT-2 (INDX-STOR) = PSR-NOT MOVE PSR-AK-FORWT
        TO STOR-IDENT-2 (INDX-STOR).
DELETE-PART-5.
    PERFORM DELETE-RPT.
    IF TEM-KEY-2 = TEM-KEY-5 NEXT SENTENCE ELSE GO TO
        DELETE-PART-4.
        ADD 1 TO DELETE-SUB.
        IF DELETE-SUB IS GREATER THAN 5 GO TO TAPE-INPUT ELSE
            GO TO DELETE-PROC.
    GET-AK.
        MOVE DELETE-PSR-NO (DELETE-SUB) TO PSR.
        COMPUTE B-C = (((PSR - BIAS) * 12) + 504) / 504.
        MOVE BLK TO NUM-BLK.
        MOVE CHAR TO NUM-CHAR.
        MOVE NUMBER-REC TO NUM-KEY.
        READ NUMBER-FILE INVALID KEY PERFORM PROGRAM-BUG.
        MOVE NUMBER-REC TO PSR-KEY, TEM-KEY-2.
        MOVE ZEROES TO NUMBER-REC.
        WRITE NUMBER-REC INVALID KEY PERFORM PROGRAM-BUG.
    GET-PSR.
        READ PSR-FILE INTO PSR-FILE-RECT INVALID KEY PERFORM
            PROGRAM-BUG.
        MOVE SPACES TO PSR-FILE-REC.
        MOVE EMPTY-PSR-FILE TO NEXT-EMPTY
        MOVE TEM-KEY-2 TO EMPTY-PSR-FILE
        WRITE PSR-FILE-HEADER INVALID KEY PERFORM PROGRAM-BUG.
    DELETE-RPT SECTION.
    DELETE-REPORT.
        INITIATE DELETE-REPORT-1.
        GENERATE REPORTED-DELETE.
        IF OP-SYS-1T = SPACES NEXT SENTENCE ELSE MOVE 1 TO OP-CTR
            MOVE ≠MASTER≠ TO SYS-OP (OP-CTR) MOVE VER-1T TO
            NO-VER (OP-CTR) GENERATE OP-SYS-RPT.
        IF OP-SYS-2T = SPACES NEXT SENTENCE ELSE MOVE 2 TO OP-CTR
            MOVE ≠MSOS≠ TO SYS-OP (OP-CTR) MOVE VER-2T TO
            NO-VER (OP-CTR) GENERATE OP-SYS-RPT.
        IF OP-SYS-3T = SPACES NEXT SENTENCE ELSE MOVE 3 TO OP-CTR
            MOVE ≠RTS≠ TO SYS-OP (OP-CTR) MOVE VER-3T TO
            NO-VER(OP-CTR) GENERATE OP-SYS-RPT.
        GENERATE RPT-OPG.
        IF OUT-DTDT = SPACES GENERATE NOT-ANS GO TO RPT-PT1
            ELSE GENERATE ANSWERED.
        IF SYS-AT = ZEROES NEXT SENTENCE ELSE MOVE 1 TO OP-CTR
            MOVE ≠MASTER≠ TO SYS-OP (OP-CTR) MOVE SYS-AT TO
            NO-VER (OP-CTR) MOVE VER-AT TO PROD-VER (OP-CTR)
            GENERATE OPER-SYS-RPT.
        IF  SYS-BT = ZEROES NEXT SENTENCE ELSE MOVE 2 TO OP-CTR
            MOVE ≠MSOS≠ TO SYS-OP (OP-CTR) MOVE SYS-BT TO
            NO-VER (OP-CTR) MOVE VER-BT TO PROD-VER (OP-CTR)
            GENERATE OPER-SYS-RPT.
        IF SYS-CT = ZEROES NEXT SENTENCE ELSE MOVE 3 TO OP-CTR
        MOVE ≠RTS≠ TO SYS-OP (OP-CTR) MOVE SYS-CT TO
        NO-VER (OP-CTR) MOVE VER-CT TO PROD-VER (OP-CTR)
        GENERATE OPER-SYS-RPT.
RPT-PT1.
    GENERATE REL-PSRS.
    MOVE 1 TO DESC-CTR.
    GENERATE DESCRIP-RPT.
    MOVE 2 TO DESC-CTR.
    GENERATE DESCRIP-RPT.
    MOVE 3 TO DESC-CTR.
    GENERATE DESCRIP-RPT.
    MOVE 4 TO DESC-CTR.
    GENERATE DESCRIP-RPT.
    MOVE 5 TO DESC-CTR.
    GENERATE DESCRIP-RPT.
    IF REPT-HEADING = ≠PSR DELETE REPORT≠ GENERATE DEL-REASON
    ELSE NEXT SENTENCE.
RPT-PT2.
    TERMINATE DELETE-REPORT-1.
DELETE-LT SECTION.
DELETE-LIST.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    MOVE HEADER-RECORD TO HEADER-RECORDT.
    MOVE PRIM-KEY TO  TEM-KEY-1.
    MOVE BACKWARD-LINKT TO PRIM-KEY.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    MOVE FORWARD-LINKT TO FORWARD-LINK.
    WRITE HEADER-RECORD INVALID KEY PERFORM PROGRAM-BUG.
    MOVE FORWARD-LINKT TO PRIM-KEY.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    MOVE BACKWARD-LINKT TO BACKWARD-LINK.
    WRITE HEADER-RECORD INVALID KEY PERFORM PROGRAM-BUG.
    MOVE SPACES TO HEADER-RECORD.
    MOVE EMPTY-HEAD-REC TO EMPTY-SAK.
    MOVE TEM-KEY-1 TO PRIM-KEY, EMPTY-HEAD-REC.
    WRITE EMPTY-RECORD INVALID KEY PERFORM PROGRAM-BUG.
LEVEL-PROC SECTION.
LEVEL-PROCESSOR.
    MOVE PLH-SAK TO PRIM-KEY.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    MOVE FORWARD-LINK TO PRIM-KEY.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    IF PRODUCT-NAME = PRODUCT-LEVEL GO TO OP-SYSTEM-CHECK.
    DISPLAY STARS, DIAGNOSTIC-L2, STARS, REC-L UPON OUT.
    GO TO TAPE-INPUT.
```

```
OP-SYSTEM-CHECK.
    IF OP-SYS-LEVEL = ≠MASTER≠ MOVE MASTER-SAK TO PRIM-KEY
        GO TO COMPLETE-LEVEL.
    IF OP-SYS-LEVEL = ≠MSOS≠ MOVE MSOS-SAK TO PRIM-KEY
        GO TO COMPLETE-LEVEL.
    IF OP-SYS-LEVEL = ≠RTS≠ MOVE RTS-SAK TO PRIM-KEY
        GO TO COMPLETE-LEVEL.
    DISPLAY STARS, DIAGNOSTIC-L1, STARS, REC-L UPON OUT.
    GO TO TAPE-INPUT.
COMPLETE-LEVEL.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    MOVE SUPPORT-LEVEL TO SUPP-LEVEL.
    WRITE HEADER-RECORD INVALID KEY PERFORM PROGRAM-BUG.
    GO TO TAPE-INPUT.
REPORT-PROCESS SECTION.
REPORT-PROCESSOR.
    IF OS-FIELD2 IS NUMERIC NEXT SENTENCE ELSE GO TO
        REPORT-PROCESSOR-1.
    IF OS-FIELD3 IS EQUAL TO SPACE MOVE OS-FIELD2 TO PSR ELSE
        MOVE OS-FIELD1 TO PSR.
    COMPUTE B-C = (((PSR - BIAS) * 12) + 504) / 504.
    MOVE BLK TO NUM-BLK.
    MOVE CHAR TO NUM-CHAR.
    MOVE NUMBER-REC TO NUM-KEY.
    READ NUMBER-FILE INVALID KEY PERFORM PROGRAM-BUG.
    MOVE NUMBER-REC TO PSR-KEY.
    READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
    MOVE ≠SINGLE PSR REPORT≠ TO REPT-HEADING.
    MOVE PSR-FILE-REC TO PSR-FILE-RECT.
    PERFORM DELETE-RPT.
    GO TO TAPE-INPUT.
REPORT-PROCESSOR-1.
    IF OS-FIELD = ≠ALL≠ PERFORM PROD-CHECK VARYING OS-NUMBER
    FROM 1 BY 1 UNTIL OS-NUMBER = 3 GO TO TAPE-INPUT.
    IF OS-FIELD = ≠MASTER≠ MOVE 1 TO OS-NUMBER.
    IF OS-FIELD = ≠MSOS≠ MOVE 2 TO OS-NUMBER.
    IF OS-FIELD = ≠RTS≠ MOVE 3 TO OS-NUMBER.
    PERFORM PROD-CHECK GO TO TAPE-INPUT.
PROD-CHECK SECTION.
PROD-CHEK.
    MOVE PLH-SAK TO PRIM-KEY, TEM-KEY-1.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    MOVE FORWARD-LINK TO TEM-KEY-2, PRIM-KEY.
    IF PROD-FIELD = ≠ALL≠ PERFORM ALL-PROD GO TO PROD-CHECK3.
PROD-CHECK1.
```

```
           READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
           MOVE FORWARD-LINK TO TEM-KEY-3.
           IF PROD-FIELD = PRODUCT-NAME GO TO PROD-CHECK2.
           IF FORWARD-LINK = TEM-KEY-1 GO TO NONE-THERE.
           GO TO PROD-CHECK1.
       PROD-CHECK2.
           MOVE PRIMARY-RECORD TO PRIMARY-RECORDT.
           IF R-SAKT (CS-NUMBER) = 0 GO TO NONE-THERE.
           MOVE 1 TO DELETE-SUB.
           MOVE R-SAKT (CS-NUMBER) TO PRIM-KEY, TEM-KEY-4.
           PERFORM FOLL-LIST.
       PROD-CHECK3.
           EXIT.
       FOLL-LIST SECTION.
       FOL-LIST.
           READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
           MOVE SECONDARY-RECORD TO SECONDARY-RECORDT.
           MOVE PSR-SAKT TO PSR-KEY.
           READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
           MOVE PSR-FILE-REC TO PSR-FILE-RECT.
           IF WHICH-FIELD = ≠ALL≠ NEXT SENTENCE ELSE IF OUT-DTDT IS
               NOT EQUAL TO SPACES AND WHICH-FIELD = ≠UNANSWERED≠
               MOVE FORWARD-LKT TO PRIM-KEY IF FORWARD-LKT = TEM-KEY-4
               GO TO PERF-EXIT ELSE GO TO FOL-LIST ELSE IF
               OUT-DTDT IS EQUAL TO SPACES AND WHICH-FIELD = ≠ANSWERED≠
               MOVE FORWARD-LKT TO PRIM-KEY IF FORWARD-LKT = TEM-KEY-4
               GO TO PERF-EXIT ELSE GO TO FOL-LIST.
           SUBTRACT AB-DIT FROM TODAYS-DATE GIVING DIFF-DATE.
           IF AGE-FIELD = ≠ALL≠ NEXT SENTENCE ELSE IF TODAYS-DATE
               IS LESS THAN 30 MOVE FORWARD-LKT TO PRIM-KEY
               IF FORWARD-LKT = TEM-KEY-4 GO TO PERF-EXIT
               ELSE GO TO FOL-LIST.
           MOVE ≠   PSR REPORT    ≠ TO REPT-HEADING.
           IF QUANTITY-FIELD = ≠DETAIL≠ PERFORM DELETE-RPT GO TO ENCK.
           IF QUANTITY-FIELD = ≠BRIEF≠ PERFORM DELETE-REPORT GO TO ENCK.
           IF QUANTITY-FIELD = ≠LIST≠ PERFORM LIST-RPT.
       ENCK.
           MOVE FORWARD-LKT TO PRIM-KEY.
           IF FORWARD-LKT = TEM-KEY-4 NEXT SENTENCE ELSE GO TO
               FOL-LIST.
           IF QUANTITY-FIELD = ≠LIST≠ AND DELETE-SUB IS GREATER THAN 1
               GENERATE LISTING-REPORT TERMINATE LIST-PSR-NUM-REPORT.
       PERF-EXIT.
           EXIT.
       LIST-RPT SECTION.
```

```
LIST-PSR-NUM.
    INITIATE LIST-PSR-NUM-REPORT.
    MOVE PSR-NUMBERT TO LIST-AREA (DELETE-SUB).
    IF DELETE-SUB = 6 GENERATE LISTING-REPORT MOVE 1 TO
        DELETE-SUB.
    ADD 1 TO DELETE-SUB.
LIST-EXIT.
    EXIT.
ALL-PROD SECTION.
ALL-PROD-PARA.
    IF FORWARD-LINK = TEM-KEY-1 GO TO ALL-PROD-EXIT.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    MOVE FORWARD-LINK TO TEM-KEY-3, PRIM-KEY.
    IF R-SAKT (OS-NUMBER) = 0 GO TO ALL-PROD-PARA.
    MOVE PRIMARY-RECORD TO PRIMARY-RECORDT.
    MOVE 1 TO DELETE-SUB.
    MOVE R-SAKT (OS-NUMBER) TO PRIM-KEY, TEM-KEY-4.
    PERFORM FOLL-LIST.
    MOVE TEM-KEY-3 TO PRIM-KEY, FORWARD-LINK.
    GO TO ALL-PROD-PARA.
ALL-PROD-EXIT.
    EXIT.
NONE-THERE SECTION.
NEIN-THERE.
    IF OS-NUMBER = 1 DISPLAY ≠ MASTER ≠ PRODUCT-NAME
    ≠ NONE THERE≠ UPON OUT.
    IF OS-NUMBER = 2 DISPLAY ≠ MSOS ≠ PRODUCT-NAME ≠ NONE THERE ≠
    UPON OUT.
    IF OS-NUMBER = 3 DISPLAY ≠ RTS ≠ PRODUCT-NAME ≠ NONE THERE ≠
    UPON OUT.
    GO TO PROD-CHECK3.
GET-EMPTY-LIST SECTION.
GET-LIST.
    MOVE PRIM-KEY TO TEM-KEY-4.
    MOVE EMPTY-HEAD-REC TO READY-LIST, PRIM-KEY.
    MOVE  HEADER-RECORD TO TEM-HOLD-1.
    READ  PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    IF FORWARD-LINK = ZERO  PERFORM  EXPAND-LIST-FILE.
    MOVE FORWARD-LINK TO EMPTY-HEAD-REC.
    MOVE  TEM-KEY-4  TO PRIM-KEY.
    MOVE  TEM-HOLD-1 TO HEADER-RECORD.
GET-EMPTY-PSR SECTION.
GET-PSR-E.
    MOVE EMPTY-PSR-FILE TO READY-PSR, PSR-KEY.
    READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
```

```
        IF NEXT-EMPTY = ZERO PERFORM EXPAND-PSR-FILE.
        MOVE NEXT-EMPTY TO EMPTY-PSR-FILE.
    EXPAND-LIST-FILE  SECTION.
    EXP-LIST.
        CLOSE PRIM-SEC-LIST.
        MOVE ZERO TO TEM-KEY-8.
        ENTER COMPASS, EXPAND, PRIM-SEC-LIST, TEM-KEY-8.
        COMPUTE PRIM-EXPAND = ((TEM-KEY-8 * 100000) + PRIM-EXPAND).
        OPEN I-O  PRIM-SEC-LIST.
        PERFORM PRIM-KEY-UPDATE.
        MOVE UP-KEY TO PRIM-KEY, TEM-KEY-5.
        PERFORM LIST-FILE-EMPTY.
        MOVE  TEM-KEY-5 TO FORWARD-LINK.
    EXPAND-PSR-FILE SECTION.
    EXP-LIST.
        CLOSE PSR-FILE.
        MOVE ZERO TO TEM-KEY-8.
        ENTER COMPASS, EXPAND, PSR-FILE, TEM-KEY-8.
        COMPUTE PSR-EXPAND = ((TEM-KEY-8 * 100000) + PSR-EXPAND).
        OPEN I-O PSR-FILE.
        PERFORM PSR-KEY-UPDATE.
        MOVE UP-KEY TO PSR-KEY, TEM-KEY-5.
        PERFORM PSR-FILE-EMPTY.
        MOVE TEM-KEY-5 TO NEXT-EMPTY.
    EXPAND-NUM-FILE SECTION.
    EXP-NUM.
        CLOSE NUMBER-FILE.
        ENTER COMPASS, EXPAND, NUMBER-FILE, TEM-KEY-8.
        OPEN I-O NUMBER-FILE.
        MOVE NUM-KEY TO TEM-KEY-6.
        MOVE NUM-EXPAND TO NUM-KEY-1.
        MOVE  ZERO TO NUM-KEY-2.
        ADD TEM-KEY-8 TO NUM-EXPAND.
        MOVE ZERO TO NUMBER-REC.
        PERFORM NUM-FILE-ZERO.
        MOVE  TEM-KEY-6 TO NUM-KEY.
    LIST-FILE-EMPTY SECTION.
    LIST-EMPTY.
        PERFORM PRIM-KEY-UPDATE.
        IF  UP-KEY  =  PRIM-EXPAND  GO TO LIST-EMPTY-END.
        MOVE UP-KEY TO FORWARD-LINK.
        WRITE HEADER-RECORD INVALID KEY PERFORM PROGRAM-BUG.
        MOVE UP-KEY TO PRIM-KEY.
        GO TO LIST-EMPTY.
    LIST-EMPTY-END.
    MOVE  ZERO TO FORWARD-LINK.
    WRITE HEADER-RECORD INVALID KEY PERFORM  PROGRAM-BUG.
PSR-FILE-EMPTY SECTION.
PSR-EMPTY.
    PERFORM PSR-KEY-UPDATE.
    IF UP-KEY = PSR-EXPAND GO TO PSR-EMPTY-END.
    MOVE UP-KEY TO NEXT-EMPTY.
    WRITE PSR-FILE-HEADER INVALID KEY PERFORM PROGRAM-BUG.
    MOVE UP-KEY TO PSR-KEY.
    GO TO PSR-EMPTY.
PSR-EMPTY-END.
    MOVE ZERO TO NEXT-EMPTY.
    WRITE PSR-FILE-HEADER INVALID KEY PERFORM PROGRAM-BUG.
NUM-FILE-ZERO SECTION.
NUM-ZERO.
    WRITE NUMBER-REC INVALID KEY PERFORM PROGRAM-BUG.
    PERFORM NUM-KEY-UPDATE.
    IF NUM-KEY-1 NOT EQUAL TO NUM-EXPAND  GO TO NUM-ZERO.
UPDATE-KEYS SECTION.
PRIM-KEY-UPDATE.
    ADD 82 PRIM-KEY GIVING UP-KEY.
    IF  U-CH GREATER THAN  1887
        ADD 1 TO U-BLK
        MOVE ZERO TO U-CH.
PSR-KEY-UPDATE.
    ADD 100000 PSR-KEY GIVING UP-KEY.
        NOTE PARAGRAPH PERFORM TO ALLOW EASY MODIFICATION
            OF BLOCKING PARAMETER.
NUM-KEY-UPDATE.
    ADD  12  TO  NUM-KEY.
    IF  NUM-KEY-2  GREATER THAN  493
        ADD 1 TO NUM-KEY-1
        MOVE ZERO TO NUM-KEY-2.
P-KEY-UPDATE.
    ADD 82 P-KEY GIVING UP-KEY.
    IF  U-CH GREATER THAN 1887
        ADD 1 TO  U-BLK
        MOVE ZERO TO U-CH.
CARD-SEQ-ERR SECTION.
CARD-SEQ-ERROR.
    MOVE PSR-NUM TO HOLD-PSR.
CD-SEQ-ERR.
    READ SORT-OUT AT END GO TO LAST-CARD.
    IF PSR-NUM = HOLD-PSR AND CODE-ALPHA = ALPHA-SAVE GO TO
        CD-SEQ-ERR.
```

```
     MOVE CODE-ALPHA TO ALPHA-SAVE
     GO TO TAPE-CHECK.
 PROGRAM-BUG SECTION.
 SYSTEM-ERROR.
     DISPLAY  ≠IRREGULAR CONDITION OCCURRED, FILE INTIGRITY QUESTI
-        ≠ONABLE≠.
     DISPLAY ≠-IRREGULAR CONDITION OCCURRED, FILE INTIGRITY QUESTI
-        ≠ONABLE≠
         UPON OUT.
     ENTER COMPASS, PRGABORT.
 INPUT-PROCEDURE SECTION.
 PARA-ONE-TIME.
     READ INFILE-CARD AT END GO TO ONE-TIME-ALSO.
     IF LEGAL-CODE RELEASE SORT-FILE-REC FROM INREC
         GO TO PARA-ONE-TIME.
     DISPLAY STARS, DIAGNOSTIC-C, STARS, INREC UPON OUT.
     GO TO PARA-ONE-TIME.
 ONE-TIME-ALSO.  EXIT.
 READ-ROUTINE SECTION.
 READ-ROUT.
     READ SORT-OUT AT END GO TO READ-ROUT-2.
     IF CODE-ALPHA = ALPHA-SAVE GO TO READ-ROUT-END.
 READ-ROUT-1.
     GO TO ADD-END, CHANGE-END, TAPE-CHECK DEPENDING ON
         GO-KEY.
 READ-ROUT-2.
     MOVE ≠Z≠ TO CODE-ALPHA.
     GO TO READ-ROUT-1.
 READ-ROUT-END.
     EXIT.
 UTILITY SECTION.
 U-PROCESSOR.
     IF U-FIELD  =   ≠LOAD FILES≠         GO TO  LOAD.
     IF U-FIELD  =   ≠DUMP FILES≠         GO TO  DUMP.
     IF U-FIELD  =   ≠COLLECT FILES≠      GO TO  COLLECT.
     IF U-FIELD  =   ≠ESTABLISH FILES≠    GO TO  ESTABLISH.
     IF U-FIELD  =   ≠LIST FILES≠         GO TO  FILE-LIST.
     DISPLAY STARS, DIAGNOSTIC-U, STARS, REC-U.
 U-PROC-1.
     READ SORT-OUT AT END
         GO TO ALTER-STOP.
     IF CODE-ALPHA =  ≠$≠  GO TO U-PROCESSOR.
     MOVE CODE-ALPHA TO ALPHA-SAVE.
     GO TO FILE-OPEN.
 LOAD SECTION.
```

```
LD-B1.
    PERFORM OPERATOR-CHECK.
    ENTER COMPASS. RLEASE, PRIM-SEC-LIST, PSR-FILE, NUMBER-FILE.
    ENTER COMPASS, LOAD, PRIM-SEC-LIST, PSR-FILE, NUMBER-FILE.
    GO TO U-PROC-1.
DUMP SECTION.
DP-1.
    ALTER ALTER-STOP TO PROCEED TO DUMP-FILES.
    GO TO U-PROC-1.
    NOTE FILES ARE DUMPED ONLY AT END OF RUN.
DUMP-FILES.
    OPEN INPUT PRIM-SEC-LIST, PSR-FILE, NUMBER-FILE.
    ENTER COMPASS, DUMP, PRIM-SEC-LIST, PSR-FILE, NUMBER-FILE.
    CLOSE PRIM-SEC-LIST, PSR-FILE, NUMBER-FILE.
    GO TO STOP-PAR.
FILE-LIST SECTION.
FILE-L1.
    OPEN INPUT PRIM-SEC-LIST. PSR-FILE, NUMBER-FILE.
    MOVE SYSTEM-DATE TO DATE-STORE.
    MOVE SYSTEM-TIME TO TIME-STORE.
    DISPLAY
        ≠1 PSR LIST FILE LISTING.  DATE  ≠  DATE-STORE
        ≠.     TIME  ≠  TIME-STORE  ≠.≠
        UPON OUT.
    DISPLAY  ≠0≠  UPON OUT.
    MOVE  BK1-CHO  TO  NUM-KEY.
FILE-L2.
    READ NUMBER-FILE INVALID KEY GO TO FILE-L3.
    DISPLAY  SPACE NUM-KEY SPACE NUMBER-REC UPON OUT.
    PERFORM NUM-KEY-UPDATE.
    GO TO FILE-L2.
FILE-L3.
    CLOSE NUMBER-FILE.
    MOVE SYSTEM-TIME TO TIME-STORE.
    MOVE SYSTEM-DATE TO DATE-STORE.
    DISPLAY
        ≠1  LIST FILE LISTING.       DATE  ≠  DATE-STORE
        ≠.     TIME  ≠  TIME-STORE  ≠.≠
        UPON OUT.
    DISPLAY  ≠0≠ UPON OUT.
    MOVE BK1-CHO TO PRIM-KEY.
FILE-L4.
    READ PRIM-SEC-LIST INVALID KEY GO TO FILE-L5.
    DISPLAY SPACE PRIM-KEY SPACE HEADER-RECORD UPON OUT.
    PERFORM PRIM-KEY-UPDATE.
```

```
           MOVE UP-KEY TO PRIM-KEY.
           GO TO FILE-L4.
       FILE-L5.
           CLOSE PRIM-SEC-LIST.
           MOVE SYSTEM-DATE TO DATE-STORE.
           MOVE SYSTEM-TIME TO TIME-STORE.
           DISPLAY
               ≠1  PSR FILE LISTING          DATE  ≠  DATE-STORE
               ≠.    TIME  ≠  TIME-STORE  ≠.≠
               UPON OUT.
           MOVE BK1-CHO TO PSR-KEY.
       FILE-L6.
           READ PSR-FILE INVALID KEY GO TO FILE-L7.
           DISPLAY  ≠0≠ PSR-KEY SPACE PSR-FILE1-MOVE PSR-FILE2-MOVE
               UPON OUT.
           DISPLAY  ≠                DESCRIPTIONS DELETED.≠ UPON OUT.
           DISPLAY  ≠                ≠ PSR-FILE3-MOVE UPON OUT.
           PERFORM PSR-KEY-UPDATE.
           MOVE UP-KEY TO PSR-KEY.
           GO TO FILE-L6.
       FILE-L7.
           CLOSE PSR-FILE.
           DISPLAY ≠1≠ UPON OUT.
           GO TO U-PROC-1.
       OPERATOR-CHECK SECTION.
       OPER-CK.
           DISPLAY ≠ RESPOND OK IF FILES ARE TO BE RELEASED≠.
           ACCEPT REL-FILES.
           IF REL-FILES IS NOT EQUAL TO ≠OK≠ DISPLAY SPACE REC-U
               SPACE ≠ OPERATOR DROP≠ UPON OUT STOP RUN.
       ESTABLISH SECTION.
       ESTAB-1.
           PERFORM OPERATOR-CHECK.
           ENTER COMPASS, RLEASE, PRIM-SEC-LIST, PSR-FILE, NUMBER-FILE.
           ENTER COMPASS, ALLOCATE,
               PRIM-SEC-LIST, +300,
               PSR-FILE, +100,
               NUMBER-FILE, +100.
           OPEN OUTPUT PRIM-SEC-LIST, PSR-FILE, NUMBER-FILE.
           MOVE BK1-CHO TO PRIM-KEY, PSR-KEY, NUM-KEY.
           MOVE  U-BIAS TO BIAS.
           MOVE 101 TO NUM-EXPAND.
           WRITE  NUMBER-REC  FROM  NUMBER-FILE-STORAGE
               INVALID  KEY   PERFORM  PROGRAM-BUG.
           PERFORM  NUM-KEY-UPDATE.
```

```
            PERFORM  NUM-FILE-ZERO.
            CLOSE  NUMBER-FILE.
            MOVE  ≠ PSR FILE HEADER≠  TO P-F-H-S-H.
            PERFORM  PSR-KEY-UPDATE.
            MOVE UP-KEY TO  EMPTY-PSR-FILE, FIRST-PSR-ENT.
            MOVE   10100000 TO  PSR-EXPAND.
            WRITE PSR-FILE-REC FROM PSR-FILE-HEADER-STORAGE
                INVALID KEY  PERFORM  PROGRAM-BUG.
            MOVE  SPACES TO PSR-FILE-HEADER-STORAGE.
            MOVE  UP-KEY TO  PSR-KEY.
            PERFORM  PSR-FILE-EMPTY.
            CLOSE PSR-FILE.
            MOVE  ≠ LIST FILE HEADER≠ TO  H-R-S.
            MOVE PRIM-KEY TO  H-R-S-SAK.
            PERFORM PRIM-KEY-UPDATE.
            MOVE  UP-KEY TO PLH-SAK, PRIM-KEY.
            PERFORM PRIM-KEY-UPDATE.
            MOVE  UP-KEY TO  EMPTY-HEAD-REC.
            MOVE  30100000  TO PRIM-EXPAND.
            MOVE  BK1-CHO TO PRIM-KEY.
            WRITE HEADER-RECORD FROM HEADER-RECORD-STORAGE
                INVALID KEY PERFORM PROGRAM-BUG.
            MOVE  PLH-SAK TO PRIM-KEY.  EMPTY-HEAD-REC, H-R-S-SAK.
            MOVE  ZERO TO PRIM-EXPAND.
            MOVE ≠ PRIMARY PRODUCT LIST≠ TO H-R-S.
            WRITE HEADER-RECORD FROM HEADER-RECORD-STORAGE
                INVALID KEY PERFORM PROGRAM-BUG.
            MOVE SPACES TO HEADER-RECORD.
            PERFORM PRIM-KEY-UPDATE.
            MOVE  UP-KEY TO PRIM-KEY.
            MOVE  30100000 TO PRIM-EXPAND.
            PERFORM LIST-FILE-EMPTY.
            CLOSE PRIM-SEC-LIST.
            GO  TO U-PROC-1.
        COLLECT SECTION.
        COLL-BEGIN.
            OPEN I-O  PRIM-SEC-LIST, PSR-FILE, NUMBER-FILE.
            MOVE  BK1-CHO TO  PRIM-KEY, P-KEY, NUM-KEY.
            READ PRIM-SEC-LIST INTO HEADER-RECORD-STORAGE
                INVALID KEY PERFORM PROGRAM-BUG.
            READ NUMBER-FILE INTO NUMBER-FILE-STORAGE INVALID KEY PERFORM
                PROGRAM-BUG
            MOVE PRIM-EXPAND TO UP-KEY.
            SUBTRACT 1 FROM U-BLK.
            SUBTRACT 1 FROM NUM-EXPAND GIVING BLK.
```

```
    ENTER COMPASS, ALLOCATE, P-S-LIST, U-BLK,
        N-FILE, BLK.
    OPEN I-O  P-S-LIST.
    OPEN OUTPUT N-FILE.
COLL-PRIMARY.
    MOVE  PLH-SAK TO PRIM-KEY.
    PERFORM P-KEY-UPDATE.
    MOVE UP-KEY TO PLH-SAK.
    WRITE T-REC-2   FROM  HEADER-RECORD-STORAGE
        INVALID KEY PERFORM PROGRAM-BUG.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    IF FORWARD-LINK = BACKWARD-LINK AND HEADER-SAK
    GO TO COLL-NUM-END.
    MOVE FORWARD-LINK TO PRIM-KEY.
    MOVE BACKWARD-LINK TO TEM-KEY-1.
    MOVE UP-KEY TO P-KEY, HEADER-SAK, TEM-KEY-2
    PERFORM P-KEY-UPDATE.
    MOVE UP-KEY TO FORWARD-LINK.
    WRITE T-REC-2   FROM PRIMARY-RECORD
        INVALID KEY PERFORM PROGRAM-BUG.
COLL-PRIM-1.
    IF  PRIM-KEY = TEM-KEY-1 GO TO  COLL-PRIM-2.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    MOVE FORWARD-LINK TO PRIM-KEY.
    MOVE P-KEY TO BACKWARD-LINK.
    MOVE TEM-KEY-2 TO HEADER-SAK.
    MOVE UP-KEY TO P-KEY.
    PERFORM P-KEY-UPDATE.
    MOVE UP-KEY TO FORWARD-LINK.
    WRITE T-REC-2  FROM PRIMARY-RECORD
        INVALID KEY PERFORM PROGRAM-BUG.
    GO TO COLL-PRIM-1.
COLL-PRIM-2.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    MOVE P-KEY TO BACKWARD-LINK.
    MOVE TEM-KEY-2 TO FORWARD-LINK, HEADER-SAK.
    MOVE UP-KEY TO P-KEY, TEM-KEY-7.
    WRITE T-REC-2   FROM PRIMARY-RECORD
        INVALID KEY PERFORM PROGRAM-BUG.
    MOVE TEM-KEY-2 TO P-KEY.
    READ  P-S-LIST INTO PRIMARY-RECORD
        INVALID KEY PERFORM PROGRAM-BUG.
    MOVE TEM-KEY-7 TO BACKWARD-LINK.
    WRITE T-REC-2   FROM PRIMARY-RECORD
        INVALID KEY PERFORM PROGRAM-BUG.
```

```
               MOVE FORWARD-LINK TO P-KEY, TEM-KEY-6.
               MOVE HEADER-SAK TO TEM-KEY-2.
           COLL-SEC-1.
               IF  P-KEY = TEM-KEY-2  GO TO COLL-SEC-DONE.
               READ P-S-LIST  INTO PRIMARY-RECORDT
                   INVALID KEY PERFORM PROGRAM-BUG.
               IF  MASTER-SAKT = ZERO  GO TO COLL-SEC-2.
               MOVE 1 TO I.
               MOVE TEM-KEY-7 TO P-KEY.
               PERFORM P-KEY-UPDATE.
               MOVE  MASTER-SAKT TO PRIM-KEY.
               MOVE UP-KEY TO MASTER-SAKT.
               PERFORM LIST-COLLECT.
           COLL-SEC-2.
               IF  MSOS-SAKT = ZERO GO TO COLL-SEC-3.
               MOVE 2 TO I.
               MOVE TEM-KEY-7 TO P-KEY.
               PERFORM P-KEY-UPDATE.
               MOVE MSOS-SAKT TO PRIM-KEY.
               MOVE UP-KEY TO MSOS-SAKT.
               PERFORM  LIST-COLLECT.
           COLL-SEC-3.
               IF  RTS-SAKT = ZERO GO TO COLL-SEC-4.
               MOVE 3 TO I.
               MOVE TEM-KEY-7 TO P-KEY.
               PERFORM P-KEY-UPDATE.
               MOVE RTS-SAKT TO PRIM-KEY.
               MOVE UP-KEY TO RTS-SAKT.
               PERFORM LIST-COLLECT.
           COLL-SEC-4.
               MOVE TEM-KEY-6 TO P-KEY.
               WRITE T-REC-2  FROM PRIMARY-RECORDT
                   INVALID KEY PERFORM PROGRAM-BUG.
               MOVE FOR-LINKT TO P-KEY, TEM-KEY-6.
               GO TO COLL-SEC-1.
           COLL-SEC-DONE.
               MOVE TEM-KEY-7 TO P-KEY.
               PERFORM P-KEY-UPDATE.
               MOVE UP-KEY TO EMPTY-HEAD-REC.
               MOVE BK1-CHO TO PRIM-KEY.
               WRITE HEADER-RECORD FROM HEADER-RECORD-STORAGE
               INVALID KEY PERFORM PROGRAM-BUG.
               MOVE PLH-SAK TO P-KEY.
           COLL-SEC-DONE-1.
               IF  P-KEY  = EMPTY-HEAD-REC GO TO COLL-SEC-DONE-2.
           PERFORM PRIM-KEY-UPDATE.
           MOVE UP-KEY TO PRIM-KEY.
           READ P-S-LIST INTO PRIMARY-RECORD
               INVALID KEY PERFORM PROGRAM-BUG.
           PERFORM P-KEY-UPDATE.
           MOVE UP-KEY TO P-KEY.
           GO TO COLL-SEC-DONE-1.
       COLL-SEC-DONE-2.
           MOVE SPACES TO HEADER-RECORD.
           PERFORM LIST-FILE-EMPTY.
           CLOSE PRIM-SEC-LIST, P-S-LIST.
           IF  U-BIAS NOT EQUAL TO SPACES GO TO COLL-NUM-END.
           WRITE T-REC-1   FROM NUMBER-FILE-STORAGE
               INVALID KEY PERFORM PROGRAM-BUG.
       COLL-NUM-1.
           PERFORM NUM-KEY-UPDATE.
           READ NUMBER-FILE INVALID KEY GO TO COLL-NUM-END.
           IF  NUMBER-REC NOT = ZERO GO TO COLL-NUM-2.
           ADD 1 TO BIAS.
           GO TO COLL-NUM-1.
       COLL-NUM-2.
           WRITE T-REC-1   FROM NUMBER-REC
               INVALID KEY PERFORM PROGRAM-BUG.
           IF  NUM-KEY-1 = NUM-EXPAND GO TO COLL-NUM-3.
           PERFORM NUM-KEY-UPDATE.
           READ NUMBER-FILE INVALID KEY PERFORM PROGRAM-BUG.
           GO TO COLL-NUM-2.
       COLL-NUM-3.
           MOVE ZERO TO NUMBER-REC.
           WRITE  T-REC-1   FROM NUMBER-REC
               INVALID KEY GO TO COLL-NUM-4.
           GO TO COLL-NUM-3.
       COLL-NUM-4.
           CLOSE N-FILE.
           OPEN INPUT N-FILE.
           READ N-FILE  AT END      PERFORM PROGRAM-BUG.
           MOVE BK1-CHO TO NUM-KEY.
           WRITE NUMBER-REC FROM NUMBER-FILE-STORAGE
               INVALID KEY PERFORM PROGRAM-BUG.
       COLL-NUM-5.
           PERFORM NUM-KEY-UPDATE.
           IF  NUM-KEY  = NUM-EXPAND GO TO COLL-NUM-END.
           READ N-FILE  AT END PERFORM PROGRAM-BUG.
           GO TO COLL-NUM-5.
       COLL-NUM-END.
           CLOSE NUMBER-FILE, N-FILE.
```

```
    ENTER COMPASS, RLEASE, P-S-LIST, N-FILE.
    GO TO U-PROC-1.
LIST-COLLECT SECTION.
LC-1.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    MOVE TEM-KEY-6 TO HEADER-LINK.
    IF  FORWARD-LINK  = BACKWARD-LINK AND HEADER-SAK
        MOVE UP-KEY TO FORWARD-LINK, BACKWARD-LINK,
           HEADER-SAK, P-KEY, TEM-KEY-7  GO TO LC-E.
    MOVE BACKWARD-LINK TO TEM-KEY-5.
    MOVE UP-KEY TO P-KEY,  HEADER-SAK,  TEM-KEY-4.
    PERFORM  P-KEY-UPDATE.
    MOVE FORWARD-LINK TO  PRIM-KEY.
    MOVE  UP-KEY TO FORWARD-LINK.
    WRITE  T-REC-2    FROM SECONDARY-RECORD
        INVALID KEY  PERFORM PROGRAM-BUG.
LC-2.
    IF  PRIM-KEY = TEM-KEY-5  GO TO LC-3.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    MOVE FORWARD-LINK TO PRIM-KEY.
    MOVE P-KEY TO BACKWARD-LINK.
    MOVE  UP-KEY TO P-KEY.
    PERFORM P-KEY-UPDATE.
    MOVE UP-KEY TO FORWARD-LINK.
    MOVE TEM-KEY-4 TO HEADER-SAK.
    WRITE  T-REC-2   FROM SECONDARY-RECORD
        INVALID KEY PERFORM PROGRAM-BUG.
    MOVE PSR-SAK TO PSR-KEY.
    READ PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
    MOVE P-KEY   TO OP-KEY  (I).
    WRITE PSR-FILE-REC  INVALID KEY PERFORM PROGRAM-BUG.
    GO TO LC-2.
LC-3.
    READ PRIM-SEC-LIST INVALID KEY PERFORM PROGRAM-BUG.
    MOVE TEM-KEY-4 TO FORWARD-LINK, HEADER-SAK.
    MOVE P-KEY TO BACKWARD-LINK.
    MOVE UP-KEY TO P-KEY, TEM-KEY-7.
    WRITE  T-REC-2   FROM SECONDARY-RECORD
        INVALID KEY PERFORM PROGRAM-BUG.
    MOVE  PSR-SAK TO PSR-KEY.
    READ  PSR-FILE INVALID KEY PERFORM PROGRAM-BUG.
    MOVE  P-KEY  TO OP-KEY  (I).
    WRITE PSR-FILE-REC INVALID KEY PERFORM PROGRAM-BUG.
    MOVE  TEM-KEY-4 TO P-KEY.
    READ  P-S-LIST  INTO SECONDARY-RECORD
```

```
        INVALID KEY PERFORM PROGRAM-BUG.
    MOVE TEM-KEY-7 TO BACKWARD-LINK.
LC-E.
    WRITE T-REC-2   FROM SECONDARY-RECORD
        INVALID  KEY  PERFORM PROGRAM-BUG.
END PROGRAM.
```

THE COBOL LANGUAGE B

COBOL CHARACTER SET

The characters recognized by COBOL include the letters of the alphabet, digits, and those characters, commonly called symbols, which are used in expressions, relations and editing. The complete computer character set consists of the 63 characters listed in the COBOL collating sequence (appendix D).

Character Set for Words

Letters A-Z, digits 0-9, and hyphen (-)

Character Set for Punctuation

,	comma
;	semicolon
.	period
"	quotation mark
(	left parenthesis
)	right parenthesis
	space

Character Set for Arithmetic Operators

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

Character Set for Relational Operators

>	greater than
<	less than
=	equal to

Character Set for Editing

B	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppress
*	check protect
$	currency sign
,	comma (decimal point)
.	period (decimal point)
/	slash

WORDS

A word is composed of a combination of not more than 30 characters chosen from the character set for words. The word cannot begin nor end with a hyphen. The space character is not allowed in a word; the space is a word separator. Wherever a space is used, more than one may be used. A word is ended by a space, or by a period, right parenthesis, comma, or semicolon followed by a space. A space must not immediately follow a left parenthesis or a beginning quotation mark. A space must not immediately precede a right parenthesis or an ending quotation mark except when the space is included in a literal.

Words are specified by the user or they are COBOL reserved words. User-defined words include all names assigned by the user to elements in the program; they must never be from the set of COBOL reserved words.

Literals are also considered user-defined words, but they are not restricted to the limits imposed on other words.

Data Name

A data name is a word containing at least one alphabetic character. It names a data item in the Data Division (file names, library names, mnemonic names, report names are all formed like data names; their functions are defined in context.)

Examples:

QUANTITY-ON-HAND	MESSAGE
LAST-YEAR-PROFIT	100A
ITEM-NUMBER	FILE-1

Condition Name

A condition name is assigned to a value, set of values or range of values, within the complete set of values that a data item may assume. The condition name must contain at least one alphabetic character. Each condition name must be unique, or be made unique through qualification. The associated data item may be the qualifier for any of its condition names. If references to this data item require indexing, subscripting or qualification, references to any of its condition names require the same combination of indexing, subscripting or qualification.

Example: VAR. name Grade. [illegible] or value of grade the condition names are set T or F

```
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.
```

A condition name may be used in conditions as an abbreviation for the relation condition. For example: IF GRADE-ONE ...

Procedure Name

A procedure name identifies a paragraph or a section in the Procedure Division. A procedure name may be composed solely of numeric characters. Two numeric procedure names are equivalent only if they are composed of the same number of digits and have the same value; 0023 is not equivalent to 23.

Examples:

```
PROC-1.     A-INPUT SECTION.
002.        100 SECTION.
```

Identifier

An identifier is a data name followed, as required, by the syntactically correct combination of qualifiers, subscripts, or indexes necessary to make unique reference to a data item.

Examples:

DAY OF MASTER-DATE

FIRST IN GRADE-ONE

MALE-FEMALE (2,5,1)

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.

Literal

A literal is a string of characters. Literals may be numeric or non-numeric.

Non-Numeric Literal

A non-numeric literal is a string of any characters allowable in the computer's character set (including reserved words but excluding the quotation mark) and bounded by quotation marks. The value of a non-numeric literal is the string of characters itself, excluding the quotation marks. Any spaces included between the quotation marks are part of the non-numeric literal and, therefore, are part of the value. All non-numeric literals are classed as alphanumeric. They may contain from 1 to 120 characters excluding the quotes.

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.

Numeric Literal

A numeric literal is a string of characters chosen from digits 0 thru 9, the plus sign, the minus sign, and the decimal point. Its value is the algebraic quantity represented by the characters in the literal. Every numeric literal is in the numeric category.

Rules for forming numeric literals:

- It must contain at least one and not more than 18 digits.
- It can contain only one sign character. If a sign is used, it must appear as the leftmost character of the literal. An unsigned literal is positive.
- It can contain only one decimal point. The decimal point is treated as an assumed decimal point, and may appear anywhere in the literal except as the rightmost character. A literal without a decimal point is an integer.

If a literal conforms to the rules for forming a numeric literal, but is enclosed in quotation marks, it is a non-numeric literal.

Examples of numeric literals:

```
15067893251459
-12572.6
+25675
1435.89
```

Floating Point Literal

A floating point literal is a numeric literal written in the following form.

[±]coefficient E[±] exponent

Rules for floating-point literals:

- The coefficient may have from one to eleven digits in the range of $0 \leq n \leq 2^{36}-1$ and a decimal point.
- The coefficient is followed immediately by the symbol E, followed by an optional plus or minus sign, followed by one to three numeric digits which indicate the power of the exponent. A zero exponent may be written as 0, 00, or 000. An unsigned exponent is assumed to be positive. The exponent is to the base 10 and may range from 0 through 308.
- The plus and minus signs preceding the coefficient and exponent are optional. All other elements of the format are required.
- A floating point literal must appear as a continuous string of characters with no intervening spaces.

Examples

15.2E5	+2.725E-6
12.4E10	-4.0E+12
-.425E101	-5.124E-20
+30E-101	405E+305
60.5E0	.68719476735E308

Reserved Words

Reserved words defined in appendix C are used for syntactical purposes and may not appear as user-defined words. Reserved words are used as key words, optional words, and connectives.

Key Word

A word that is required when it appears in a source program format is called a key word. Within each format, key words are uppercase and underlined. All verbs, for example, are key words which must be included to designate the operation to be performed.

Optional Words

Uppercase words in a format that are not underlined are called optional words since they may appear or not as the user chooses. The presence or absence of each optional word within a format does not affect the compiler's translation. An optional word must not be misspelled or replaced by another word not explicitly stated to be its equivalent.

Special Registers

The word TALLY is the name of a special register implicitly described as a five-digit integer. TALLY is used primarily to hold information produced by the EXAMINE statement. TALLY may also be used as a data name wherever an elementary data item of integral value may appear.

The special registers SYSTEM-DATE and SYSTEM-TIME are restricted for use as the subjects of MOVE statements in the Procedure Division and the objects of SOURCE clauses in the Report Section of the Data Division; they may not be used as literals. In a MOVE statement, the receiving field must have a PICTURE of X(8). Likewise, in the Report Section, the elementary item description containing the SOURCE clause must have a PICTURE clause of X(8).

SYSTEM-DATE represents the current date in the form mm/dd/yy: mm = month, dd = day, yy = year. SYSTEM-TIME represents the time of day when the object program begins execution. Its format is hh/mm/ss: hh = hour, mm = minute, ss = second.

Figurative Constants

Certain constants, called figurative constants, have been assigned fixed data names. Figurative constants must not be bounded by quotation marks, or they become non-numeric literals. The singular and plural forms of figurative constants are equivalent and may be used interchangeably.

Figurative Constant	Meaning
ZERO ZEROS ZEROES	Represents the value 0, or one or more of the character 0, depending on context.
SPACE SPACES	Represents one or more blanks or spaces.
HIGH-VALUE HIGH-VALUES	Represents one or more of the character with the highest value in the COBOL collating sequence.
LOW-VALUE LOW-VALUES	Represents one or more of the character with the lowest value in the COBOL collating sequence.
QUOTE QUOTES	Represents one or more of the character " or the character substituted for it on computers that do not use a quotation mark. The word QUOTE cannot be used in place of a quotation mark in a source program to bound a non-numeric literal.
ALL literal	Represents one or more of the string of characters comprising the literal. The literal must be either a non-numeric literal or a figurative constant other than ALL literal. When a figurative constant is used, the word ALL is redundant and may be used for readability only.

When a figurative constant represents a string of one or more characters, the length of the string is set by the compiler according to the following rules:

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

A figurative constant can be used any place where a literal appears in the format. When the literal is restricted to numeric characters, ZERO (ZEROS) (ZEROES) is the only figurative constant permitted.

Examples:

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" IS EQUAL TO COUNT-FIELD...	Assuming COUNT-FIELD has 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.

Connectives

There are three types of connectives:

- Qualifiers used to associate a data name or paragraph name with its qualifier: OF, IN.
- A series connective that links two or more consecutive operands: , (comma) or two or more consecutive clauses: ; (semicolon).
- Logical connectives used in the formation of conditions: AND, OR, AND NOT, OR NOT.

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 the optional words IS and THAN are used or not.

PUNCTUATION

The punctuation symbols, period, comma, and semicolon, are shown within the formats. All division and section headers are followed by a period with the remainder of the line blank. Specific rules for each division follow:

Identification Division: Commas and semicolons may be used for readability within comment paragraphs; they are not shown in the formats. Each paragraph and paragraph name must be terminated by a period followed by a space.

Environment Division: Commas and semicolons shown in the formats are optional. Each paragraph and paragraph name must terminate with a period followed by a space.

Data Division: Commas and semicolons shown in the formats are optional. Each file and data description entry must be terminated by a period.

Procedure Division: Commas shown in the formats are optional. A semicolon may be used between statements in a sentence. In addition, the semicolon may appear in an IF statement immediately following the condition and also immediately preceding the keyword ELSE. Each sentence must be terminated by a period and a space. Procedure names are terminated by a period and one or more spaces.

When a period, semicolon, or comma is used, it immediately follows a word and must be immediately followed by a space. A beginning quotation mark is not followed by a space nor is an ending quotation mark preceded by a space unless the space is part of the nonnumeric literal. A left parenthesis is followed by a space only when followed by + - (; in these cases a space is required. A right parenthesis is preceded by a space only when it is preceded by a). A right parenthesis is always followed by a space. Parentheses must be paired.

COBOL RESERVED WORD LIST

ACCEPT	CF	CURRENCY	EXAMINE
ACCESS	CH	DATA	EXIT
ACCESS-PRIVACY	CHARACTERS	DATE-COMPILED	FD
ACTUAL	CLOSE	DATE-WRITTEN	FILE
ADD	COBOL	DE	FILE-CONTROL
ADDRESS	CODE	DECIMAL	FILE-LIMIT
ADVANCING	COLUMN	DECIMAL-POINT	FILE-LIMITS
AFTER	COMMA	DECLARATIVES	FILLER
ALL	COMMON-STORAGE	DENSITY	FINAL
ALPHABETIC	COMP	DEPENDING	FINIS
ALPHANUMERIC	COMP-1	DESCENDING	FIRST
ALTER	COMP-2	DETAIL	FOOTING
ALTERNATE	COMPASS	DISK	FOR
AND	COMPUTATIONAL	DISPLAY	FORTRAN
ARE	COMPUTATIONAL-1	DIVIDE	FROM
AREA	COMPUTATIONAL-2	DIVISION	GENERATE
AREAS	COMPUTE	DOWN	GIVING
ASCENDING	CONFIGURATION	EDITION-NUMBER	GO
ASSIGN	CONSOLE	ELSE	GREATER
AT	CONTAINS	END	GROUP
AUTHOR	CONTIGUOUS	ENDING	HEADING
BEFORE	CONTROL	ENTER	HIGH
BEGINNING	CONTROLS	ENVIRONMENT	HIGH-VALUE
BINARY	COPY	EQUAL	HIGH-VALUES
BLANK	CORR	EQUALS	HOLD
BLOCK	CORRESPONDING	ERROR	HYPER
BY	CRT	EVERY	I-O

I-O-CONTROL	LOW-VALUES	PIC	REPORTS
ID	MEMORY	PICTURE	RERUN
IDENTIFICATION	MODE	PLUS	RESERVE
IF	MODIFICATION-PRIVACY	POSITION	RESET
IN	MODULES	POSITIVE	RETENTION-CYCLE
INDEX	MOVE	PRINTER	RETURN
INDEXED	MULTIPLE	PROCEDURE	REVERSED
INDICATE	MULTIPLY	PROCEED	REWIND
INITIATE	NEGATIVE	PROCESS	RF
INPUT	NEXT	PROCESSING	RH
INPUT-OUTPUT	NO	PROGRAM-ID	RIGHT
INSTALLATION	NOT	PUNCH	ROUNDED
INTO	NOTE	QUOTE	RUN
INVALID	NUMBER	QUOTES	SA
IS	NUMERIC	RANDOM	SAME
JUST	OBJECT-COMPUTER	RD	SCRATCH
JUSTIFIED	OCCURS	READ	SD
KEY	OF	READER	SEARCH
KEYS	OFF	RECORD	SECTION
LABEL	OMITTED	RECORDING	SECTOR
LAST	ON	RECORDS	SECURITY
LEADING	OPEN	RECORD-MARK	SEEK
LEFT	OPTIONAL	REDEFINES	SEGMENT-LIMIT
LESS	OR	REEL	SEGMENTED
LIMIT	OUTPUT	REEL-NUMBER	SELECT
LIMITS	OWNER	RELEASE	SENTENCE
LINE	OWNER-ID	REMARKS	SEQUENCED
LINE-COUNTER	PAGE	RENAMES	SEQUENTIAL
LINES	PAGE-COUNTER	RENAMING	SET
LOCK	PERFORM	REPLACING	SIGN
LOW	PF	REPORT	SIZE
LOW-VALUE	PH	REPORTING	SORT

SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STATUS
STOP
SUBTRACT
SUM
SYNC
SYNCHRONIZED
SYSTEM-DATE
SYSTEM-INPUT
SYSTEM-OUTPUT
SYSTEM-PUNCH
SYSTEM-TIME
TALLY
TALLYING
TAPE
TERMINATE
THAN
THEN
THROUGH } equivalent
THRU }
TIMES
TO
TRACE
TRACK
TTY
TYPE
UNEQUAL
UNIT
UNTIL
UP
UPON
USAGE
USE
USING
VALUE
VALUES
VARYING
WHEN
WITH
WORDS
WORKING-STORAGE
WRITE
ZERO
ZEROES
ZEROS

COBOL COLLATING SEQUENCE D

The following table shows the relationship between characters of the COBOL set and their equivalent machine, printer, and punched card codes. They are listed according to ascending collating sequence. The COBOL quote (") character is represented as a not-equal-to (≠) sign on the printer. The 4-8 multiple punch must be used to represent the quote in a COBOL source card.

Characters shown with an asterisk are not available in COBOL source language but would be treated in the sequence indicated if present in data.

Collating Sequence	Internal Code	External Code	Printer Character	Cards	
				Character	Punch
00	60	20	Δ	Δ	blank
01	15	15	≤*		8,5
02	16	16	%*		8,6
03	17	17	[*		8,7
04	75	35	→ *		0,8,5
05	76	36	≡*		0,8,6
06	77	37	∧ *		0,8,7
07	55	55	↑*		11,8,5
08	56	56	↓*		11,8,6
09	57	57	>		11,8,7
10	35	75	≥*		12,8,5
11	36	76	¬		12,8,6
12	33	73	.	.	12,8,3
13	34	74	)	)	12,8,4
14	37	77	;		12,8,7
15	20	60	+	+	12
16	53	53	$	$	11,8,3
17	54	54	*	*	11,8,4
18	40	40	-	-	11
19	61	21	/	/	0,1

Collating Sequence	Internal Code	External Code	Printer Character	Cards	
				Character	Punch
20	73	33	,	,	0,8,3
21	74	34	(	(	0,8,4
22	13	13	=	=	8,3
23	14	14	≠	(dash)	8,4
24	32	72	<	+0*	12,0
25	21	61	A	A	12,1
26	22	62	B	B	12,2
27	23	63	C	C	12,3
28	24	64	D	D	12,4
29	25	65	E	E	12,5
30	26	66	F	F	12,6
31	27	67	G	G	12,7
32	30	70	H	H	12,8
33	31	71	I	I	12,9
34	52	52	∨*	-0*	11,0
35	41	41	J	J	11,1
36	42	42	K	K	11,2
37	43	43	L	L	11,3
38	44	44	M	M	11,4
39	45	45	N	N	11,5
40	46	46	O	O	11,6
41	47	47	P	P	11,7
42	50	50	Q	Q	11,8
43	51	51	R	R	11,9
44	72	32	]*		0,8,2
45	62	22	S	S	0,2
46	63	23	T	T	0,3
47	64	24	U	U	0,4
48	65	25	V	V	0,5

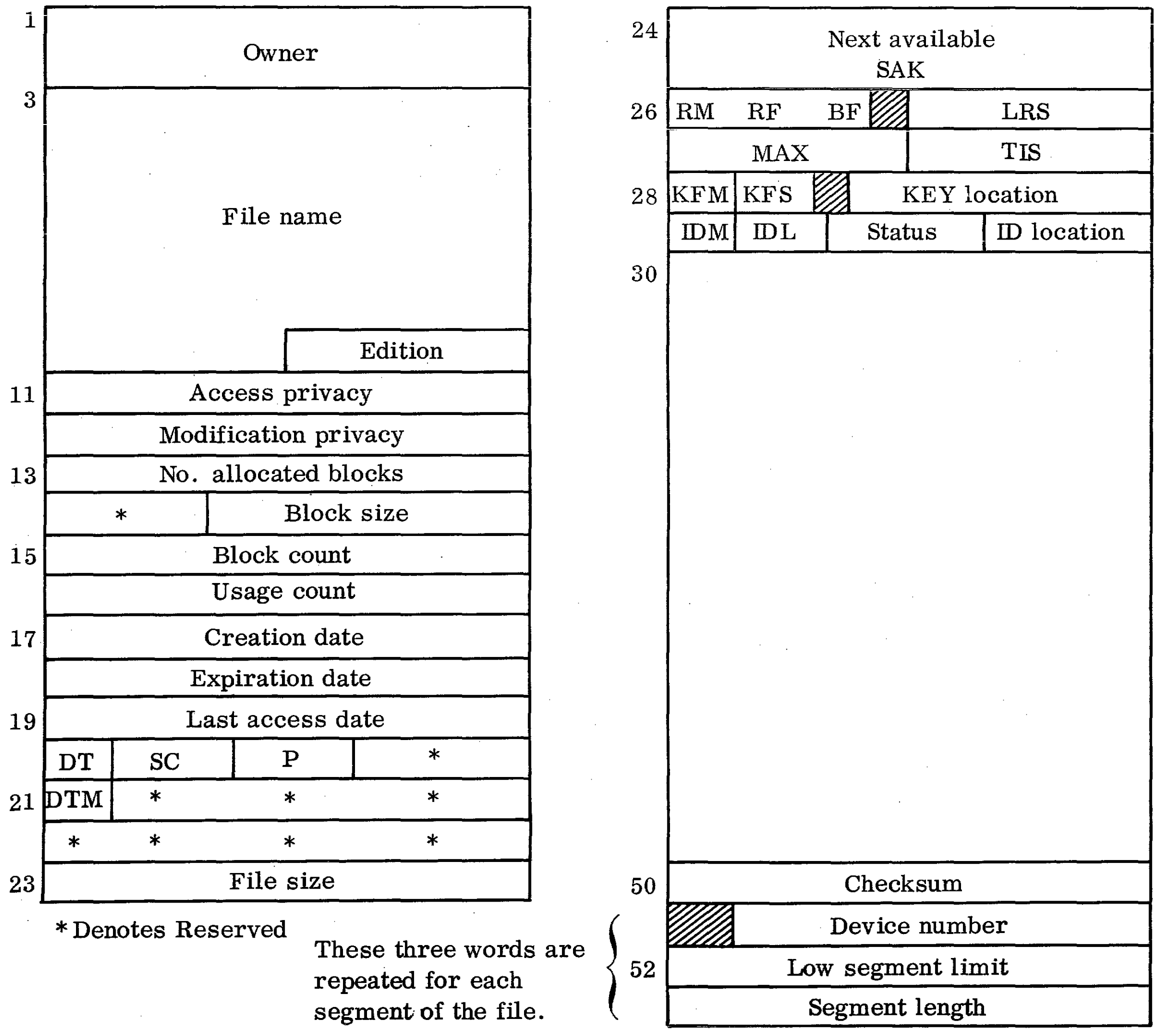
Collating Sequence	Internal Code	External Code	Printer Character	Cards	
				Character	Punch
49	66	26	W	W	0,6
50	67	27	X	X	0,7
51	70	30	Y	Y	0,8
52	71	31	Z	Z	0,9
53 †		00	:*		
53 †	00	12	0	0	0
54	01	01	1	1	1
55	02	02	2	2	2
56	03	03	3	3	3
57	04	04	4	4	4
58	05	05	5	5	5
59	06	06	6	6	6
60	07	07	7	7	7
61	10	10	8	8	8
62	11	11	9	9	9

† Within the COBOL Collating sequence, external codes of 00 and 12 are the same.

STANDARD FILE LABELS E

This appendix illustrates the standard file labels used with COBOL systems. The mass storage file label is maintained by MASTER, and cannot be accessed by the user. The field lengths specified in this appendix for standard label fields should be observed in the VALUE OF phrase associated with the LABEL RECORDS ARE STANDARD clause.

<u>MASS STORAGE FILE LABEL</u>



FIELD NAME	SIZE	DESCRIPTION
File Identifier	40 characters	Uniquely identifies a file in label directory. The standard identifier consists of: Owner Identification - 8 characters File Name - 30 characters Edition Number - 2 characters The 40-character field may be divided in other ways at installation option.
Access Privacy	4 characters	Supplied when file is allocated and must be supplied for each succeeding OPEN request.
Modification Privacy	4 characters	Supplied when file is allocated and must be supplied for each RELEASE, EXPAND, and MODIFY request.
No. Blocks Allocated	24 bits	Binary integer giving the number of blocks allocated to the file.
Block Size	18 bits	Binary integer indicating number of 6-bit characters in each record block ($0 < \text{block size} < 131072$).
Block Count	24 bits	Binary integer, highest block number written. If file is processed sequentially, this is the number of blocks written into the file ($0 \leq \text{block count} < 2^{23}$).
Usage Count	24 bits	Binary count of number of times file has been opened.
Creation Date	24 bits	Date in the form yymmdd, stored as a binary integer, supplied by I/O system when file is allocated.
Expiration Date	24 bits	Date in the form yymmdd, stored as a binary integer, supplied by user when the file is allocated. This field determines when a file may be deleted.
Last Access Date	24 bits	Date in the form yymmdd, stored as a binary integer, supplied by I/O system each time file is opened or changed.

FIELD NAME	SIZE	DESCRIPTION
DT (device type)	1 character	6-bit code to indicate type of mass storage device: DT = 40_8 (1311 Disk Packs) = 41_8 (852 Disk Packs) = 50_8 (853 Disk Packs) = 51_8 (854 Disk Packs) = 60_8 (813, 814 Disk Files) = 70_8 (863 Drum)
SC (segment count)	6 bits	Binary integer giving number of segments in file ($0 < SC < 64$).
P (protection)	1 character	Protection flags for use by I/O system. Values currently defined: 0 File may be read or written 1 File may not be written
DTM (device type modifier)	1 character	6-bit code which provides further device information. For 1311 and 852 disk packs, the values are: XXXXX0 Track Mode XXXXX1 Sector Mode For 853, 854, 813, 814, and 863, the value is: XXXXX1 Sector Mode
File Size	24 bits	Binary integer indicating number of allocatable units (tracks) assigned to file ($0 <$ file size $< 2^{23}$).
Next Available SAK	8 characters	File block and record positions where next record can be written.
RM (record mark)	1 character	Character which terminates each record when the record format is record mark variability.

FIELD NAME	SIZE	DESCRIPTION
RF (record format)	3 bits	Denotes the type of file records: 00 Fixed length records 01 Key field contains total number of characters 02 Key field contains the number of occurrences of a fixed length trailer item 03 Universal format 04 Record mark specified by RM terminates each record
BF (block format)	1 bit	1 = One logical record per block 0 = Logical records are blocked and each block contains two word header which specifies next block number (NBN) and position of first available character in block (POFAC).
LRS (logical record size)	2 characters	Logical record size, in characters, of fixed length record; size of fixed portion of variable records with trailer items. Zero if records vary by key field or record mark.
MAX (maximum logical record size)	2 characters	Maximum size in characters of variable portion of logical records. For variable records with trailers, this is size of trailer item times maximum number of occurrences. For all others this is maximum size of record within file.
TIS (trailer item size)	2 characters	Trailer item size in characters if RF = 02, else zero.
KFM (key field mode)	1 bit	Mode of key field address 0 Key field is within each record 1 Key field is outside record (not in file)
KFS (key field size)	5 bits	Number of characters in key field.
Key Location	17 bits	Character position of key field relative to beginning of record if key field is within record.

FIELD NAME	SIZE	DESCRIPTION
IDM (ID Mode)	1 bit	Type of record identification associated with every record in file. 0 = alphanumeric 1 = numeric
IDL (ID Length)	5 bits	Length in characters of record identification field.
Status	1 character	Reserved to reflect current status of the file as defined by each operating system or library program.
ID Location	2 characters	Starting character position of identification field in each record of the file.
Checksum	24 bits	24-bit binary checksum of entire label. This field is checked by I/O system to detect accidental modification of label.
Device Number	3 characters	Number of device on which this file segment is stored. This field is checked against device label to insure that proper packs are mounted.
Low Segment Limit	24 bits	Binary hardware address at which this file segment begins.
Segment Length	4 characters	Number of allocatable units (tracks) in this segment.
*(reserved)	89 characters	These fields are reserved for future use of I/O system.

MAGNETIC TAPE

GENERAL HEADER LABELS

All header label records are 80 characters (480 bits) long, and are unblocked. They are recorded in even parity (BCD) at the same density as the remainder of the data file. Header records are separated from succeeding data records by an interrecord gap only. Header label record fields are defined below; they are positioned as shown within the physical record. Values that may be used within these fields are also indicated.

Field Name	Starting Character Position	Length in Characters	Defined Values BCD Characters Only	Function
Density†	1	1	2, 5, 8	Specifies density of recording file
Header Label Identifier	2	2	( )	Identifies record as header label record
Not used	4	2		
Retention Code	6	3	000-999	Specifies, in days, file retention period
File Name†	9	14	Any combination of legal BCD characters	Identifies file
Reel Number†	23	2	01-99	Sequence of reels for multireel files
Date Written	25	6	Any legal numeric date, expressed as mmddyy	Date written is used with retention period to determine release date of file
Edition Number†	31	2	00-99 or blank	Identifies a single file set
User Supplied Information	33	48	Any combination of legal BCD characters	User comments field

† Used by *DEF open unit function

MAGNETIC TAPE

STANDARD ENDING LABELS

The input-output control system writes ending labels for all files with standard labeling and at the end of each reel of a multireel file.

Field Name	Starting Character Position	Length in Characters	Defined Values BCD Characters Only	Function
End-of-Tape or End-of-File	1	3	EOT or EOF	End of tape for intermediate reel; End of file for final tape
Block Count	4	5	numeric	Physical record (block) count for reel if multireel file; for file if multifile or single file reel
User supplied information	9	72	Any combination of legal BCD characters	User comments field

FILE BLOCKING FORMATS

The input-output system blocks and deblocks user files in accordance with information supplied in the BLOCK CONTAINS and RECORD CONTAINS clauses of the FD entry for each file. Physical record formats depend on the hardware type on which the file resides.

The maximum size of a block and logical record is 131,067 characters.

The following chart shows the possible variations of BLOCK and RECORD clauses. The reference codes used in the chart are defined in the following pages.

File Blocking Reference Chart

BLOCK CONTAINS \ RECORD CONTAINS	clause omitted	integer-3 CHARACTERS	integer-3 TO integer-4 CHARACTERS	integer-3 TO integer-4 CHARACTERS DEPENDING ON RECORD MARK	integer-3 TO integer-4 CHARACTERS DEPENDING ON data-name (data-name PIC 9(5))	integer-3 TO integer-4 CHARACTERS DEPENDING ON data-name (USAGE COMP-1)
clause omitted	F_1	F_1	F_2	F_2	F_2	F_3
integer-1 CHARACTERS	F_4	F_4	F_5	F_5	F_5	F_5
integer-1 RECORDS	F_6	F_6	F_7	F_7	F_7	F_7
integer-1 TO integer-2 CHARACTERS	F_8	F_8	F_8	F_8	F_8	F_8
integer-1 TO integer-2 RECORDS	F_9	F_9	F_{10}	F_{10}	F_{10}	F_{10}

Reference Code	Resultant Block Format
F_1	Unblocked, fixed-length logical records. Size of physical record is computed from record description.
F_2	Unblocked logical records. Physical record may be variable on magnetic tape, but on mass storage the record is always fixed length with size equal to integer-4.
F_3	Illegal combination. Universal records must always be blocked.
F_4	Blocked or unblocked logical records, depending on whether integer-1 is a multiple of the record description length. Size of physical record is always integer-1 characters.
F_5	Blocked logical records. Size of block is always integer-1 characters on mass storage; the block may be variable on magnetic tape.
F_6	Blocked, fixed-length logical records. Size of block is equal to integer-1 times the length of the record description.
F_7	Blocked logical records. Size of block is equal to integer-1 times integer-4 for mass storage; block may be variable on magnetic tape.
F_8	Blocked logical records. Block size is always integer-2 characters on mass storage; may be variable length on tape.
F_9	Blocked logical records. Mass storage block size is equal to integer-2 times the length of the record description; may be variable on magnetic tape.
F_{10}	Blocked logical records. Mass storage block size is equal to integer-2 times integer-4; may be variable on tape.

INPUT/OUTPUT SUMMARY TABLE G

This table indicates the various features included in COBOL for performing input/output operations on a data file. The list which follows is a guide for cross referencing the table.

Summary of I/O Options

Options	Hardware Type					
	Mass Storage		Magnetic Tape	System Files (INP, OUT, PUN)	CRT TTY	Unit Record Reader, Printer, Punch
	Permanent	Scratch				
Access Mode	S, R	S, R	S	S	S	S
Labeling	S	O	S, O, D	S, O, D	O	S, O, D
Blocking	B, U, R	B, U, R	B, U	U	U	U
Block Size	FH	FH	F, V	X	X	X
Logical Record Size	F, RM, K1, K2	F, RM, K1, K2	F, RM, K1	F	F	F
Allocation	A	A	X	X	X	X
Expansion	E	E	X	X	X	X
Release	U	A	X	X	X	X
Actual Key	AR, AS	AR, AS	X	X	X	X
Use Declaratives	SE, LB	SE	SE, LB	LB	X	LB
Open	03, 04	03, 04	01, 02, 04	04	03, 04	04
Close	C1	C1	C1, C2, C3, C4	C1	C1	C1
Read	R1, R2	R1, R2	R1	R1	R1	R1
Write	W1, W2	W1, W2	W1	W1	W1	W1
Seek	S	S	X	X	X	X

Option	Code	Meaning
ACCESS MODE	S	SEQUENTIAL
	R	RANDOM
LABELING	S	STANDARD
	O	OMITTED
	D	data name
BLOCKING	B	Blocked logical records
	U	Unblocked logical records
	R	RESPOND - UNIVERSAL blocking
BLOCK SIZE	F	Fixed Length - no block header
	FH	Fixed Length - eight character header (mass storage only)
	V	Variable length
LOGICAL RECORD SIZE	F	Fixed length
	RM	Variable depending on record mark
	K1	Variable depending on data-name (key field - BCD)
	K2	Variable depending on data name (key field - COMPUTATIONAL-1)
ALLOCATION	A	File may be allocated internally
EXPANSION	E	File may be expanded internally if it is in sequential mode.
RELEASE	A	Entire file released at end of run
	U	Unused portion released when file is closed if in segmented, sequential mode
ACTUAL KEY	AR	Actual key must be updated by user if file is random mode
	AS	I/O control will update actual key if file is sequential mode
USE DECLARATIVES	SE	Use after standard error recovery
	LB	Use before/after standard beginning/ending label procedure

Option	Code	Meaning
OPEN	01	OPEN $\left\{\begin{matrix}\text{INPUT}\\\text{OUTPUT}\end{matrix}\right\}$ WITH NO REWIND
	02	OPEN INPUT REVERSED
	03	OPEN I-O
	04	OPEN $\left\{\begin{matrix}\text{INPUT}\\\text{OUTPUT}\end{matrix}\right\}$
CLOSE	C1	CLOSE
	C2	CLOSE WITH NO REWIND
	C3	CLOSE WITH LOCK
	C4	CLOSE REEL
READ (INTO)	R1	READ; AT END (sequential)
	R2	READ; INVALID KEY (random)
WRITE (FROM)	W1	WRITE (sequential)
	W2	WRITE; INVALID KEY (random)
SEEK	S	SEEK; INVALID KEY
ILLEGAL OPTION	X	

DIFFERENCES BETWEEN MASS STORAGE COBOL AND USASI COBOL

USASI COBOL and Mass Storage (MS) COBOL both operate under control of the MASTER operating system. They are compatible except for the language differences described in this appendix.

The elements omitted from USASI COBOL either have no effect on the object program or their functions can be performed by other elements present in both versions of COBOL/MASTER. The elements omitted from MS COBOL are features in USASI COBOL which have no equivalent in MS COBOL.

Elements in MS COBOL not in USASI COBOL	Alternate Methods Available in Both
ENVIRONMENT DIVISION:	
MEMORY SIZE clause of SOURCE-COMPUTER paragraph	None, used for documentation only
DATA DIVISION:	
CONSTANT SECTION	Constants can be defined with the VALUE clause in Working-Storage and Common-Storage sections.
File Description Clauses:	
FILE CONTAINS	None, used for documentation only
NON-STANDARD option of LABEL RECORDS clause	Data-name option of LABEL RECORDS clause can be used to specify nonstandard labeling. Label checking is done by the input-output control system in USASI COBOL.
Record Description Clauses:	
SIZE	Number of symbols representing character position in PICTURE(includes all symbols but V S and P).
CLASS	Type of character in PICTURE: 9 is NUMERIC A is ALPHABETIC X is ALPHANUMERIC or AN
POINT LOCATION	V in PICTURE
SIGN or SIGNED	S in PICTURE

Elements in MS COBOL not in USASI COBOL	Alternate Methods Available in Both
Record Description Clauses (cont'd):	
ZERO SUPPRESS	Z used as replacement character in PICTURE
CHECK PROTECT	* used as replacement character in PICTURE
FLOAT DOLLAR SIGN	currency symbol ($) used as replacement character in PICTURE
PROCEDURE DIVISION:	
Conditional statement connectives or operators:	
OTHERWISE	ELSE
THEN	; (semicolon)
UNEQUAL	NOT EQUAL TO or NOT =
EQ	EQUAL TO or =
NQ	NOT EQUAL TO or NOT =
GR	GREATER or >
LS	LESS or <

Elements Common to MS and USASI COBOL

RESERVE $\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{integer-1}\end{array}\right\}$ ALTERNATE AREA

clause in FILE-CONTROL

For MS COBOL:

NO	Produces no buffering
1	Produces single buffering
2	(or clause omitted) Produces double buffering

For USASI COBOL:

NO	(or clause omitted) Produces no buffering for unblocked files
1	Produces single buffering for unblocked files and double buffering for blocked sequential files
2	Produces double buffering for blocked and unblocked sequential files

The elements given below are available in USASI COBOL but not in MS COBOL. Except for abbreviations, which can be replaced by the complete word, these elements represent new features which have no equivalent in MS COBOL.

ENVIRONMENT DIVISION:

MODULES option of OBJECT-COMPUTER paragraph. (documentation only)

Literal IS mnemonic-name clause of SPECIAL-NAMES paragraph

CURRENCY SIGN IS literal clause of SPECIAL-NAMES paragraph

DECIMAL-POINT IS COMMA clause of SPECIAL-NAMES paragraph

RENAMING clause of FILE-CONTROL paragraph

DATA DIVISION:

COMMON-STORAGE SECTION

REPORT SECTION

SD entry (sort file description)

File Description:

SECTOR/TRACK and SEGMENTED/CONTIGUOUS options of RECORDING MODE clause

VALUE OF option of LABEL RECORD IS data-name clause

REPORT clause

Record Description:

level-number 66 and RENAMES clause

INDEX and COMPUTATIONAL-2 options of USAGE clause

JUSTIFIED RIGHT clause

ASCENDING/DESCENDING KEY and INDEXED BY options of OCCURS clause

PIC abbreviation of PICTURE

COMP abbreviation of COMPUTATIONAL

COMP-1 abbreviation of COMPUTATIONAL-1

PROCEDURE DIVISION:

GIVING series in ADD, SUBTRACT, MULTIPLY and DIVIDE

TO series in ADD

BY option in DIVIDE

UNIT option in CLOSE

language-name option of ENTER

UNTIL and VARYING options of PERFORM

REVERSED option of OPEN INPUT

REPLACING and procedure-name options of COPY

BEFORE/AFTER ADVANCING option of WRITE

compound conditions joined with AND

COMPUTE verb

Sort Feature:

RELEASE verb

RETURN verb

SORT verb

Table Handling Feature:

SEARCH verb

SET verb

Source Program Debugging Feature:

TRACE verb

Report Writer Feature:

BEFORE REPORTING option of USE statement

GENERATE verb

INITIATE verb

TERMINATE verb

DIAGNOSTICS

DIAGNOSTICS

Each diagnostic generated by the compiler has the following format:

nnnnn c English text diagnostic word / D idn

nnnnn	Source program line number where error occurred.
c	Character code specifying severity of error:
	S Severe error. No object code is generated for the statement; execution and binary deck generation suppressed.
	E Probable error. Code generated for statement; however, results during execution are unpredictable.
	W Warning. A minor inconsistency encountered. Program expected to run correctly.
word	Current source text element deemed erroneous.
D idn	Internal name assigned to a data-name for PASS 2 diagnostics.

In all cases, program compilation continues in order to diagnose additional errors.

IDENTIFICATION/ENVIRONMENT DIVISION

Type	Message
W	*ACCESS MODE* CLAUSE IGNORED FOR NONMASS STORAGE FILE.
W	*ACCESS MODE* CLAUSE REQUIRED FOR MASS STORAGE--ASSUMED SEQUENTIAL.
W	*ACTUAL KEY* CLAUSE IGNORED FOR NONMASS STORAGE FILE.
S	*ACTUAL KEY* MISSING FOR RANDOM FILE--SET TO SEQUENTIAL.
S	*ASSIGN* MISSING--FILE IGNORED (word)
S	COMPUTER NAME MUST BE *3300* OR *3500* (word)
S	*DATA DIVISION* HEADING MISSING--COMPILATION TERMINATED.
S	DATA-NAME USED WITH ACTUAL KEY CLAUSE MAY NOT BE QUALIFIED.
W	DUPLICATE CLAUSE IN SELECT STATEMENT--IGNORED (word)
S	DUPLICATE *CURRENCY SIGN* CLAUSE--SECOND CLAUSE IGNORED.
S	DUPLICATE *DECIMAL POINT* CLAUSE--SECOND CLAUSE IGNORED.
S	DUPLICATE HEADER--FIRST DEFINITION ACCEPTED (word)
S	DUPLICATE SELECT STATEMENT FOR CURRENT FILE-NAME--2ND SELECT IGNORED.
W	*END REEL* IN *RERUN* CLAUSE NOT VALID FOR MASS STORAGE--ASSUMED *END UNIT*.
W	*END UNIT* IN *RERUN* CLAUSE NOT VALID FOR TAPE FILE--ASSUMED *END REEL*.

Type	Message
W	*END UNIT* INVALID IN *RERUN* CLAUSE FOR RANDOM ACCESS FILE.
W	*FILE-LIMITS* CLAUSE IGNORED FOR NON-MASS STORAGE FILE.
S	FILE NOT DEFINED VIA A *SELECT* STATEMENT (word)
W	HEADER OUT OF ORDER--IGNORED (word)
S	*IDENTIFICATION DIVISION* MISSING OR OUT OF ORDER.
S	ILLEGAL CLAUSE IN SPECIAL-NAMES SECTION (word)
S	ILLEGAL CLAUSE WITHIN SELECT STATEMENT (word)
S	ILLEGAL *DSI* IN HARDWARE ASSIGNMENT--IGNORED (word)
S	ILLEGAL DSI IN *RERUN* CLAUSE (word)
S	ILLEGAL FILE-NAME IN *RENAMING* CLAUSE--CLAUSE IGNORED (word)
S	ILLEGAL FILE-NAME IN SELECT STATEMENT--FILE IGNORED (word)
S	ILLEGAL HARDWARE-NAME (word)
S	ILLEGAL HARDWARE TYPE IN *RERUN* CLAUSE (word)
S	ILLEGAL HEADER--IGNORED (word)
S	ILLEGAL LITERAL IN *CURRENCY IS* CLAUSE (word)
S	ILLEGAL *TUI* IN HARDWARE ASSIGNMENT--IGNORED (word)
S	ILLEGAL WORD IN *ACCESS-MODE* CLAUSE (word)
S	ILLEGAL WORD IN *ACTUAL KEY* CLAUSE (word)
S	ILLEGAL WORD IN *CURRENCY IS* CLAUSE (word)
S	ILLEGAL WORD IN *DECIMAL-POINT IS COMMA* CLAUSE (word)
S	ILLEGAL WORD IN *FILE-LIMITS* CLAUSE (word)
S	ILLEGAL WORD IN *IMPLEMENTATOR-NAME IS MNEMONIC-NAME* CLAUSE (word)
S	ILLEGAL WORD IN *LITERAL IS MNEMONIC-NAME* CLAUSE (word)
S	ILLEGAL WORD IN MEMORY SIZE CLAUSE (word)
S	ILLEGAL WORD IN *MULTIPLE REEL* CLAUSE (word)
S	ILLEGAL WORD IN *RESERVE AREA* CLAUSE--IGNORED (word)
W	INTEGER IN FILE-LIMITS CLAUSE TOO LARGE--VALUE NORMALIZED TO ≤ 8388607
W	INTEGER-2 OF *RESERVE* IS GREATER THAN 2--SET TO 2.
W	INTEGER LITERAL MUST BE POSITIVE--MINUS SIGN IGNORED (word)
S	INTEGER MISSING FROM *POSITION* OPTION OF *MULTIPLE FILE* CLAUSE.
S	INVALID CLAUSE IN I-O-CONTROL PARAGRAPH (word)
S	INVALID INTEGER IN *RERUN* CLAUSE.
S	INVALID WORD IN *RERUN* CLAUSE (word)
S	LITERAL MUST BE 1 CHARACTER--LITERAL IGNORED (word)

Type	Message
S	MNEMONIC-NAME PREVIOUSLY DEFINED (word)
S	MULTIPLE DEFINITION OF FILE-NAME IN *SAME AREA* OR *MULTIPLE FILE* CLAUSES (word) A file-name mentioned in a MULTIPLE FILE clause may not be mentioned in a SAME AREA clause since MULTIPLE FILE implies sharing same areas. Likewise, a file-name may not be mentioned more than once within either clause or in multiple like clauses.
W	*MULTIPLE REEL* CLAUSE IGNORED FOR MASS STORAGE FILE.
W	*MULTIPLE REEL/UNIT* CLAUSE VALID ONLY FOR TAPE AND MASS STORAGE FILES--CLAUSE IGNORED.
W	*MULTIPLE UNIT* CLAUSE IGNORED FOR TAPE FILE.
S	NO FILE-NAMES GIVEN IN *MULTIPLE FILE* CLAUSE (word)
S	NO FILE-NAMES GIVEN IN *SAME-AREA* CLAUSE (word)
S	*OPTIONAL* DISCREPANCY BETWEEN MASTER FILE AND CURRENT FILE (word)
W	PARAGRAPH HEADER PRECEDING THE COPY STATEMENT IS EXPECTED AT THIS POINT (word)
S	*POSITION* INTEGER IN *MULTIPLE FILE* CLAUSE CANNOT BE ZERO.
S	POSITION NUMBER OF A FILE MAY NOT EXCEED 63.
S	PREVIOUS ERROR CAUSED CURRENT FILE TO BE UNDEFINED (word)
S	PROGRAM-ID NAME IS MISSING OR ILLEGAL (word)
S	RENAMED FILE PREVIOUSLY DEFINED AS A RENAMING FILE (word)
S	RENAMING FILE WAS PREVIOUSLY RENAMED (word)
S	REQUIRED HEADER MISSING OR OUT OF ORDER (word)
S	REQUIRED WORD MISSING.
S	*SELECT* MISSING--FILE IGNORED (word)
S	*SECTION* MISSING FROM SECTION HEADER.
S	TEXT MUST NOT FOLLOW CURRENT HEADER (word)
S	THE WORD *DIVISION* IS MISSING.
W	*THRU* OPTION OF FILE-LIMITS CLAUSE ILLEGAL FOR RANDOM ACCESS FILE.
S	UNDEFINED FILE-NAME IN *RERUN* CLAUSE (word)
S	UNDEFINED FILE-NAME IN *SAME AREA* OR *MULTIPLE FILE* CLAUSE (word)

DATA DIVISION

Type	Message
S	A CONDITION NAME MAY NOT BE ASSOCIATED WITH A LEVEL 66 ITEM.
S	A CONDITION NAME MAY NOT BE ASSOCIATED WITH AN ITEM WHOSE USAGE IS INDEX.
S	A REDEFINED ITEM MAY NOT CONTAIN AN OCCURS CLAUSE NOR BE SUBORDINATE TO ITEM CONTAINING ONE.
E	ACCESS-PRIVACY VALUE EXCEEDS 4 CHARACTERS.
S	ACCUMULATED SIZE OF PRECEDING GROUP ITEM IS GREATER THAN 131,067 CHARACTERS.
E	ALL SUBORDINATE ITEMS WITHIN A COMPUTATIONAL-1 GROUP MUST BE COMPUTATIONAL-1.
E	ALL SUBORDINATE ITEMS WITHIN A COMPUTATIONAL-2 GROUP MUST BE COMPUTATIONAL-2.
E	ALL SUBORDINATE ITEMS WITHIN AN INDEX GROUP MUST BE INDEX.
E	AN ITEM MAY NOT CONTAIN VALUE CLAUSE IF IT IS SUBORDINATE TO AN ITEM WITH A VALUE CLAUSE.
S	AN ITEM SUBORDINATE TO AN OCCURRING ITEM MAY NOT HAVE A VALUE CLAUSE.
S	*ASCENDING/DESCENDING KEY* CLAUSE REQUIRES AN *OCCURS* CLAUSE.
S	*BINARY, DECIMAL, SECTOR, TRACK* EXPECTED--CURRENT WORD INVALID (word)
E	*BLANK WHEN ZERO* CLAUSE MAY NOT BE USED UNLESS ITEM IS NUMERIC EDITED.
E	BLOCK CONTAINS CLAUSE INTEGER-2 MUST BE GREATER THAN INTEGER-1.
E	BLOCK CONTAINS CLAUSE MUST HAVE POSITIVE INTEGERS.
S	COMMON-STORAGE SECTION ENCOUNTERED TWICE.
E	COMMON-STORAGE SECTION MUST PRECEDE WORKING-STORAGE SECTION.
S	COMPUTATIONAL-1 LITERAL MAY NOT EXCEED + OR - 8338607.
S	COMPUTATIONAL-2 LITERAL EXPONENT MAY NOT EXCEED + OR - 308.
S	COMPUTATIONAL-2 ITEM ASSIGNED ILLEGAL LITERAL.
S	CURRENT FILE-NAME CANNOT BE A SORT FILE.
	One or more of the following clauses in the ENVIRONMENT DIVISION were used to describe the ENVIRONMENT of the current file (RENAMING, ACTUAL KEY, FILE-LIMITS, ACCESS MODE, PROCESSING MODE, RERUN ON, MULTIPLE FILE, SAME AREA), thus, the file may not be a SORT file.
W	CURRENT VALUE CLAUSE CONSIDERED DOCUMENTARY.
E	CURRENT WORD IS INVALID IN *USAGE* CLAUSE (word)
E	CURRENT WORD SHOULD BEGIN IN *A* MARGIN OF SOURCE TEXT (word)
S	DATA DIVISION MUST BEGIN WITH A SECTION HEADER
S	DATA-NAME EXPECTED--CURRENT WORD INVALID (word)
S	DATA-NAME IS NOT DEFINED IN DATA DIVISION (word)
	A data-name mentioned within a DATA DIVISION qualification string has not been defined.
S	DATA-NAME LEVEL NUMBERS MUST BE EQUAL IN REDEFINES CLAUSE.
E	*DATA RECORDS* CLAUSE MISSING IN FILE DESCRIPTION(FD).

Type	Message
E	*DATA RECORDS* CLAUSE IN SORT-FILE DESCRIPTION(SD)
E	*DIVISION* EXPECTED AT THIS POINT (word)
E	DUPLICATE CLAUSE IN ENTRY--CLAUSES OTHER THAN FIRST IGNORED.
E	EDITION-NUMBER VALUE EXCEEDS 2 CHARACTERS.
S	ELEMENT CLASS CONFLICT DETECTED. The compiler has detected either multiple PICTURE clauses for the data item or a VALUE has been assigned which does not meet the class defined in the data item PICTURE.
S	EXPONENT OF COMPUTATIONAL-2 LITERAL MUST BE 1-3 DIGITS.
S	FD LEVEL ALLOWED ONLY IN FILE SECTION.
S	FIELD NAMED IN *SEQUENCED ON* CLAUSE FOR THIS FILE IS OUTSIDE OF THE RECORD.
S	FIELD NAMED IN *SEQUENCED ON* CLAUSE FOR THIS FILE MAY NOT BE COMP-1 OR COMP-2.
S	FILE-NAME NOT SELECTED IN ENVIRONMENT DIVISION (word)
S	FILE NAME NOT UNIQUE WITHIN DATA DIVISION (word)
S	FILE SECTION ENCOUNTERED TWICE.
E	FILE SECTION MUST PRECEDE OTHER DATA DIVISION SECTIONS.
S	*FILLER* OR DATA-NAME EXPECTED--CURRENT WORD INVALID (word)
S	HIGHEST DATA-NAME QUALIFIER IS NOT UNIQUE (word) The last data-name mentioned in a qualification string must be unique when qualifying the following data items: 1. Data-name-1 of the OCCURS...DEPENDING ON clause 2. Data-name-3, data-name-4, etc. of the VALUE OF clause in a level record description.
E	ID VALUE EXCEEDS 14 CHARACTERS.
E	ID VALUE EXCEEDS 30 CHARACTERS-MASS STORAGE.
S	ILLEGAL PICTURE CLAUSE (word)
S	ILLEGAL TO RENAME ITEMS WITH LEVELS 01, 66, 77, OR 88.
S	ILLEGAL TO SPECIFY VALUE CLAUSE WITH REDEFINES CLAUSE.
E	ILLEGAL TO SYNCHRONIZE AN ITEM SUBORDINATE TO AN ITEM CONTAINING A VALUE CLAUSE.
S	IN RENAMES CLAUSE DATA-NAME-2 MUST PRECEDE DATA-NAME-3.
S	IN RENAMES CLAUSE DATA-NAME-3 MAY NOT BE SUBORDINATE TO DATA-NAME-2.
W	INCORRECT STANDARD LABEL RECORD SPECIFIED.
S	INDEX-NAME NOT UNIQUE (word)
S	INDEXED BY CLAUSE MAY NOT BE USED WITHOUT AN OCCURS CLAUSE.
S	INSUFFICIENT BLOCK SIZE.
S	INTEGER NUMBER EXPECTED--CURRENT WORD INVALID (word)

Type	Message
S	INVALID LEVEL NUMBER HIERARCHY WITHIN RECORD DESCRIPTION. Within a record description a level number has been encountered which is less than the previous level just processed, however, a prior item with an equal level number has not been defined. Example: 01 ALPHA.... 03 BETA.... 03 BETA-B.... 04 DELTA.... 02 ZETA....
E	ITEM MAY NOT HAVE USAGE OF COMP-1, COMP-2, OR INDEX IF SUBORDINATE TO ITEM WITH DISPLAY USAGE
E	ITEM MAY NOT HAVE USAGE OF COMP-1, COMP-2, OR INDEX IF SUBORDINATE TO ITEM WITH A VALUE CLAUSE
E	ITEM MAY NOT HAVE USAGE OF DISPLAY OR INDEX IF SUBORDINATE TO ITEM WITH COMPUTATIONAL USAGE.
S	ITEM MAY NOT HAVE VALUE CLAUSE IF SUBORDINATE TO AN ITEM WITH A REDEFINES CLAUSE.
E	ITEM MUST BE NUMERIC IF SUBORDINATE TO AN ITEM WITH COMPUTATIONAL USAGE.
S	ITEM WITH OCCURS DEPENDING ON CLAUSE ILLEGAL IN FILE WITH RECORD CONTAINS DEPENDING ON CLAUSE.
E	JUSTIFIED CLAUSE ILLEGAL AT GROUP LEVEL
S	KEYFIELD FROM AN OCCURS DEPENDING ON CLAUSE IN FILE SECTION MUST BE A NUMERIC ITEM $\leq$ 5 DIGITS
S	KEYFIELD FROM AN OCCURS DEPENDING ON CLAUSE MAY NOT BE SUBORDINATE TO ITEM CONTAINING CLAUSE.
S	KEYFIELD FROM AN OCCURS DEPENDING ON CLAUSE MAY NOT BE WITHIN THE VARIABLE PART OF THE RECORD.
S	KEYFIELD IN RECORD CONTAINS DEPENDING ON CLAUSE MUST BE COMP-1 OR NUMERIC ITEM OF $\leq$ 5 DIGITS.
S	*LABEL RECORDS* CLAUSE MISSING IN FILE DESCRIPTION(FD).
E	*LEFT* OR *RIGHT* EXPECTED--CURRENT WORD INVALID (word)
S	LEVEL 77 ILLEGAL IN FILE SECTION.
E	LEVEL 77 ITEMS MUST BE DEFINED FIRST IN COMMON AND WORKING STORAGE.
S	LEVEL 77 MAY NOT HAVE SUBORDINATE LEVELS.
S	LEVEL NUMBER FOLLOWING AN FD MUST BE 01.
S	LITERAL ASSOCIATED WITH A NON-NUMERIC ITEM MUST BE A QUOTED LITERAL OR FIGURATIVE CONSTANT.
S	LITERAL OR FIGURATIVE CONSTANT EXPECTED--CURRENT WORD INVALID (word)
S	LITERAL, FIGURATIVE CONSTANT OR DATA-NAME EXPECTED--CURRENT WORD INVALID (word)
S	MASS STORAGE FILE MUST HAVE STANDARD LABELS.

Type	Message
E	MODIFICATION-PRIVACY VALUE EXCEEDS 4 CHARACTERS.
S	MORE THAN 3 LEVELS OF NESTED OCCURS CLAUSES ENCOUNTERED.
S	NO RECORD DEFINED FOR THE FILE.
S	NON-USASI ELEMENT,-CLAUSE IGNORED.
S	OBJECTS OF RENAMES CLAUSE MAY NOT BE OCCURRING ITEMS NOR BE SUBORDINATE TO AN OCCURRING ITEM.
E	OCCURS CLAUSE INTEGER-2 MUST BE GREATER THAN INTEGER-1.
S	*OCCURS* CLUASE ILLEGAL AT LEVEL 01 OR 77.
E	OCCURS CLAUSE MUST USE POSITIVE INTEGERS.
S	*OMITTED* OR *STANDARD VALUE OF* OR DATA-NAME EXPECTED--CURRENT WORD INVALID (word)
E	OWNER VALUE EXCEEDS 8 CHARACTERS.
S	PICTURE CLAUSE CONFLICTS WITH CLASS OF ITEM.
E	PICTURE CLAUSE ILLEGAL AT GROUP LEVEL.
W	POINT ALIGNMENT OF LITERAL ASSIGNED TO DATA ITEM CAUSED LOSS OF NON-ZERO DIGITS ON LEFT.
W	POINT ALIGNMENT OF LITERAL ASSIGNED TO DATA ITEM CAUSED LOSS OF NON-ZERO DIGITS ON RIGHT.
S	PRECEDING DEFINITION DEFINED A STORAGE AREA OF DIFFERENT SIZE THAN THE ORIGINAL ITEM.
W	QUOTED LITERAL IS TOO LARGE FOR ITEM--RIGHT TRUNCATION OCCURRED.
E	RECORD CONTAINS CLAUSE INTEGER-2 MUST BE GREATER THAN INTEGER-1.
E	RECORD CONTAINS CLAUSE MUST HAVE POSITIVE INTEGERS.
E	RECORD CONTAINS CLAUSE NECESSARY IF USING LEVEL 01 WITH REPORTS ARE CLAUSE.
S	*RECORD-MARK* OR DATA-NAME EXPECTED--CURRENT WORD INVALID (word)
S	*RECORD* OR *RECORDS* EXPECTED--CURRENT WORD INVALID (word)
S	RECORD SIZE HAS BEEN DESCRIBED BY TWO OCCURS DEPENDING ON CLAUSES --SECOND CLAUSE IGNORED.
E	REDEFINES CLAUSE ILLEGAL AT LEVEL 66 AND 88.
S	*REDEFINES* CLAUSE MAY NOT BE SPECIFIED FOR AN OCCURRING ITEM.
E	REDEFINES CLAUSE MAY NOT BE USED AT LEVEL 01 ENTRIES IN FILE SECTION.
W	*REDEFINES* MUST IMMEDIATELY FOLLOW DATA-NAME-1.
S	REDEFINITION MUST IMMEDIATELY FOLLOW THE ENTRIES DESCRIBING THE AREA BEING REDEFINED.
E	REEL-NUMBER VALUE EXCEEDS 2 DIGITS.
S	*RENAMES* EXPECTED--CURRENT WORD INVALID (word)
S	RENAMES CLAUSE RESULTS IN ITEM SIZE GREATER THAN 32K WORDS.

Type	Message
E	RETENTION-CYCLE VALUE EXCEEDS 3 DIGITS.
E	*RIGHT* EXPECTED--CURRENT WORD INVALID (word)
S	SD LEVEL ALLOWED ONLY IN FILE SECTION.
S	SEARCH KEY IS OUTSIDE LIMITS OF TABLE AREA.
S	SEARCH KEY MAY NOT HAVE AN OCCURS CLAUSE.
S	*SECTION* EXPECTED--CURRENT WORD INVALID (word)
E	SYNCHRONIZED CLAUSE ILLEGAL AT GROUP LEVEL.
S	TABLE OVERFLOW--CANNOT EXPAND.
	The user should re-compile with more core scheduled for the job.
S	THE KEYFIELD FROM AN ACTUAL KEY CLAUSE MUST HAVE A LENGTH OF 12 CHARACTERS.
S	THIS FILE MAY NOT APPEAR IN A FD SINCE ITS SELECT STATEMENT CONTAINS A RENAMING CLAUSE.
W	TERMINAL PERIOD MISSING.
S	UNDEFINED DATA-NAME USED IN REDEFINES CLAUSE (word)
S	UNDEFINED DATA-NAME USED IN RENAMES CLAUSE (word)
S	VALUE ASSIGNED TO COMPUTATIONAL ITEM MUST BE NUMERIC LITERAL.
S	VALUE ASSIGNED TO COMPUTATIONAL-1 ITEM MUST BE NUMERIC INTEGER LITERAL.
E	VALUE CLAUSE ILLEGAL IN AN ENTRY USING OCCURS CLAUSE.
S	*VALUE OF* EXPECTED--CURRENT WORD INVALID (word)
S	*VALUE* OR *VALUES* EXPECTED--CURRENT WORD INVALID (word)
S	WORD OUT OF CONTEXT--SCANNING RESUMED AT NEXT KEYWORD (word)
S	WORKING-STORAGE SECTION ENCOUNTERED TWICE.
E	*ZERO* EXPECTED--CURRENT WORD INVALID (word)

Type	Message
S	ABSOLUTE LINE NUMBER SPECIFICATION MAY NOT FOLLOW RELATIVE LINE SPECIFICATION.
S	ABSOLUTE LINE NUMBER SPECIFIED IS NOT GREATER THAN PRECEDING LINE.
S/W	CLAUSE APPEARS MORE THAN ONCE IN SAME REPORT GROUP/ITEM DESCRIPTION.
S	*CODE* EXPECTED TO FOLLOW *WITH* (word) The word CODE must follow the word WITH in the WITH CODE mnemonic-name clause.
S	*COLUMN NUMBER* CLAUSE REQUIRED WHEN REPORT GROUP IS TO BE PRESENTED. The COLUMN NUMBER clause is required when one of the following clauses is given in the elementary item description: GROUP INDICATE, JUSTIFIED, VALUE, or BLANK WHEN ZERO.
S	CURRENT CLAUSE IS USED AT AN INVALID LEVEL NUMBER. It is illegal to use the following clauses at the RD level: LINE NUMBER, NEXT GROUP, TYPE, USAGE, COLUMN NUMBER, GROUP INDICATE, JUSTIFIED RIGHT, PICTURE, RESET, BLANK WHEN ZERO, SOURCE, SUM, and VALUE. It is illegal to give an RD level clause at the group or element level.
S	CURRENT ELEMENT IS NOT RECOGNIZABLE (word)
S	CURRENT OPERAND EXPECTED TO BE AN INTEGER (word)
S	CURRENT OPERAND EXPECTED TO BE A PROGRAMMER ASSIGNED NAME (word)
S	DATA-NAME EXPECTED TO FOLLOW *SUM----UPON* CLAUSE (word)
S	DATA-NAME IS IMPROPERLY QUALIFIED OR UNDEFINED (word)
W	*DISPLAY* MUST FOLLOW *USAGE* (word) If the USAGE clause is specified, it should be USAGE IS DISPLAY. A report writer element is always assumed to have display usage.
S	GROUP CONTROL IDENTIFIER NOT DEFINED IN CONTROL CLAUSE.
W	*GROUP INDICATE* CLAUSE MAY BE SPECIFIED ONLY FOR TYPE DETAIL ELEMENTARY ITEMS.
S	ILLEGAL GROUP NAME. The data-name following the level number of a report group must be unique within a report and cannot reflect qualification, subscripting, or indexing.
S	ILLEGAL GROUP TYPE CLAUSE (word)
S	ILLEGAL *LINE IS NEXT PAGE* CLAUSE. The NEXT PAGE option of the LINE clause is illegal in TYPE PAGE HEADING and TYPE PAGE FOOTING report groups.
S	ILLEGAL NAME IN SOURCE, SUM OR RESET CLAUSE.
S	ILLEGAL *NEXT GROUP* CLAUSE.
S	ILLEGAL QUALIFICATION OR SUBSCRIPTING OF ASSIGNED WORD (word) The current word may not be qualified or subscripted in the current context of the source code.

Type	Message
S	ILLEGAL REPORT GROUP LINE NUMBER SPECIFICATION.
	The first subordinate report item description which follows a report group description cannot contain a line number specification when the group description specifies a line number clause.
S	ILLEGAL REPORT-NAME (word)
S	ILLEGAL SUMMARY REPORTING--NO SUM COUNTERS DEFINED.
S	ILLEGAL VALUE SPECIFIED (word)
S	ILLEGAL *WITH CODE MNEMONIC-NAME* CLAUSE (word)
S	ILLEGAL VALUE IN REPORT LINE NUMBER CLAUSE (word)
S	INTEGER-1 REQUIRED IN PAGE CLAUSE WHEN ABSOLUTE SPACING USED.
S	INTEGER-3 REQUIRED IN PAGE CLAUSE WHEN PAGE HEADING DEFINED AND DETAILS USE RELATIVE SPACING.
S	INTEGER-4 REQUIRED IN PAGE CLAUSE WHEN PAGE FOOTING DEFINED AND DETAILS USE RELATIVE SPACING.
S	INVALID *COLUMN NUMBER* CLAUSE.
S	INVALID CONTROL GROUP-CONTROL IDENTIFIER PREVIOUSLY USED.
S	INVALID CONTROL GROUP-NO CONTROL CLAUSE DEFINED AT RD LEVEL.
S	INVALID DATA-NAME IN RESET CLAUSE (word)
	Data-name-1 in the RESET clause must be specified in the CONTROL clause in the report description. It must be of a higher level in the CONTROL hierarchy than the data-name associated with the report group in which the RESET clause appears.
S	INVALID *FIRST DETAIL* INTEGER (word)
S	INVALID *LAST DETAIL* INTEGER (word)
S	INVALID *NEXT GROUP* CLAUSE SPECIFICATION.
	The NEXT GROUP clause may not be used in a TYPE PAGE HEADING or TYPE PAGE FOOTING report group.
S	INVALID PAGE *FOOTING* INTEGER (word)
S	INVALID PAGE *HEADING* INTEGER (word)
S	INVALID *PAGE LIMIT* INTEGER (word)
S	INVALID PICTURE CLAUSE.
S	INVALID RESET CLAUSE--CONTROL HIERARCHY RULES VIOLATED.
S	INVALID *SUM* CLAUSE SPECIFICATION.
	The SUM clause may appear only in a TYPE CONTROL FOOTING or TYPE DETAIL report group.
S	INVALID SUM COUNTER USAGE--CONTROL HIERARCHY RULES VIOLATED.
S	LEVEL NUMBER ERROR-01 IS MISSING.
S	LINE CLAUSE ABSOLUTE VALUE EXCEEDS PAGE CLAUSE LIMITS.
S	LINE CLAUSE CONFLICTS WITH PAGE CLAUSE SPECIFICATIONS.
S	*LINE NUMBER* CLAUSE IS REQUIRED FOR REPORT LINE.

Type	Message
S	MNEMONIC-NAME IN *WITH CODE* CLAUSE IS UNDEFINED OR INVALID (word)
	Either the name is not defined or the wrong mnemonic-name in the SPECIAL-NAMES SECTION has been referenced.
S	NEXT GROUP CLAUSE ABSOLUTE VALUE EXCEEDS PAGE CLAUSE LIMITS.
S	NO LEVEL NUMBER FOLLOWING AN RD-NO REPORT GROUPS.
S	*PICTURE* CLAUSE IS REQUIRED FOR ELEMENTARY ITEM.
S	RD LEVEL INDICATOR MISSING (word)
	The compiler continues searching through the SOURCE RECORDS until a level RD is found.
S	REPORT GROUP *TYPE* CLAUSE IS REQUIRED.
S	RESET CONTROL IDENTIFIER NOT DEFINED IN CONTROL CLAUSE.
S	*SOURCE, SUM OR VALUE* CLAUSE IS REQUIRED FOR ELEMENTARY ITEM.
S	SUM CLAUSE OPERAND NOT DEFINED IN SOURCE CLAUSE OF DETAIL REPORT GROUP.
S	UNDEFINED NAME IN THE *CONTROLS ARE* CLAUSE (word)
W	UNUSED IDENTIFIER IN CONTROL CLAUSE.
S	*WITH CODE MNEMONIC-NAME* CLAUSE MUST PRECEDE *COPY* STATEMENT.
S	*ZERO* EXPECTED TO FOLLOW *BLANK WHEN* (word)

Type	Message
S	A FILE NAME MUST NOT BE REFERENCED IN MORE THAN ONE USE STATEMENT (word)
S	A RECORD NAME WAS NOT FOUND FOR THE FILE-NAME (word)
S	A SECTION-NAME MUST FOLLOW THE DECLARATIVES HEADER.
S	ADD OR SUBTRACT OPERANDS EXCEED 18 DIGITS WHEN SUPERIMPOSED ON DECIMAL POINTS.
S	*AFTER* MAY ONLY FOLLOW A CONDITION WITHIN A PERFORM STATEMENT (word)
S	ARE MORE LEFT PARENTHESES THAN RIGHT PARENTHESES IN COMPUTE STATEMENT.
S	ARE MORE RIGHT PARENTHESES THAN LEFT PARENTHESES IN COMPUTE STATEMENT.
S	*ASCENDING/DESCENDING* MUST BE SPECIFIED IN SORT STATEMENT (word)
	Either ASCENDING or DESCENDING must be specified at least once. If ON is specified, either word must immediately follow.
S	*AT END* OPTION OF READ STATEMENT VALID ONLY FOR SEQUENTIAL ACCESS FILES (word)
S	*AT END* OR *INVALID KEY* REQUIRED IN READ STATEMENT (word)
	Either AT END or INVALID KEY is required in the READ statement. If AT is specified, END must be given also.
S	*AT END* REQUIRED IN RETURN STATEMENT (word)
	If AT is specified, then END must be given also.
S	ATTEMPTED ARITHMETIC OPERATION ON NON-NUMERIC ITEM.
S	*BEFORE/AFTER ADVANCING* OPTION IS VALID ONLY FOR NON-MASS STORAGE (word)
S	*BEFORE* OR *AFTER* EXPECTED IN CURRENT WRITE STATEMENT (word)
	The current word is not recognizable. The compiler is looking for BEFORE, AFTER, INVALID, or the next verb.
S	*BEFORE* OR *AFTER* MUST FOLLOW *USE* (word)
S	*BY* EXPECTED AT THIS POINT IN *COPY REPLACING* (word)
S	*BY* IS MISSING IN EXAMINE STATEMENT (word)
	The word BY is required in the REPLACING BY option of the EXAMINE TALLYING statement and is required by itself in the EXAMINE REPLACING statement.
S	*BY* MUST FOLLOW FIRST OPERAND OF MULTIPLY STATEMENT (word)
S	*BY* MUST FOLLOW *UP* OR *DOWN* IN A SET STATEMENT (word)
S	CANNOT MIX INDEXING AND SUBSCRIPTING IN ONE REFERENCE (word)
S	COMP-1 OR COMP-2 FIELDS CAN BE COMPARED TO NUMERIC FIELD ONLY.
S	COMPILER DOES NOT ALLOW MORE THAN 63 LEFT PARENTHESES WITHOUT ANY INTERVENING RIGHT PAREN.
S	*CORRESPONDING* OPERATIONS ILLEGAL FOR ITEMS WITH LEVEL GREATER THAN 49.
S	*CORRESPONDING* OPERATIONS ILLEGAL FOR ELEMENTARY ITEMS.
S	CURRENT IDENTIFIER DOES NOT BELONG TO FILE SPECIFIED BY *SQRT* STATEMENT (word)
S	CURRENT IDENTIFIER MUST NOT BE USED IN MORE THAN ONE STATEMENT (word)

Type	Message
S	CURRENT OPERAND OF ENTER STATEMENT MUST BE AN IDENTIFIER, FILE, PROCEDURE NAME, OR LITERAL (word)
S	CURRENT OPERAND OF ENTER STATEMENT MUST BE A ROUTINE-NAME (word)
S	CURRENT WORD HAS OCCURRED OUT OF CONTEXT--COMPILER IGNORES AND CONTINUES AT NEXT VERB (word)
	The word DECLARATIVES, USE, EXIT, ELSE, or END has been used out of context. The word is ignored and the compiler looks for the next key word.
S	CURRENT WORD IN PERFORM STATEMENT SHOULD BE *TIMES* (word)
	The second element, exclusive of the THRU option, is an assigned word or numeric literal and thus implies the TIMES option is being used. The current word should be TIMES.
S	CURRENT WORD IN PERFORM STATEMENT SHOULD BE *UNTIL* OR *VARYING* (word)
	The second element, exclusive of the THRU option, is not an assigned word or numeric literal, thus UNTIL or VARYING IS EXPECTED.
S	DATA-NAME CANNOT OCCUR PRIOR TO *UP/DOWN BY* IN *SET* (word)
S	DATA-NAME HAS MORE THAN 49 LEVELS OF QUALIFICATION (word)
S	DATA-NAME IS NOT UNIQUE AND NOT QUALIFIED.
S	DATA-NAME IS NOT UNIQUE WITHIN RANGE OF PREVIOUS QUALIFIER (word)
S	DATA-NAME USED WHERE A FILE-NAME WAS EXPECTED (word)
S	DEPENDING ON OPTION OF GO TO MISSING (word)
	The DEPENDING ON option must be used when more than one procedure-name is specified in a GO TO statement.
S	DIVIDE, MULTIPLY, SUBTRACT, PERFORM OPERAND MUST BE IDENTIFIER OR NUMERIC LITERAL (word)
	The operand (word) does not meet the requirements of a programmer-assigned word.
S	*DIVISION* MUST FOLLOW *PROCEDURE* (word)
	The compiler assumes that DIVISION has been omitted and continues compilation.
S	EACH IDENTIFIER USED IN PERFORM MUST BE A NUMERIC ELEMENTARY ITEM (word)
S	EITHER *OUTPUT* OR *GIVING* OPTION REQUIRED IN SORT STATEMENT (word)
S	*ELSE* ENCOUNTERED AND CURRENT VERB IS NOT *IF* (word)
S	*ELSE* OR TERMINAL PERIOD MUST FOLLOW *NEXT SENTENCE* EXCEPT IN SEARCH (word)
	Within a conditional statement, the word ELSE or a terminal period must follow NEXT SENTENCE. If none of these were found a test is made for a key word or A-margin element. This diagnostic is produced after these tests fail to find the necessary element.
S	*END DECLARATIVES* DETECTED OUTSIDE OF DECLARATIVES SECTION (word)
S	*END DECLARATIVES* MUST BE FOLLOWED BY A PROCEDURE-NAME (word)
S	*END* DETECTED IN A-MARGIN--NEXT WORD NOT *PROGRAM* NOR *DECLARATIVES* (word)
S	*END* DETECTED WHILE PROCESSING DECLARATIVES--ASSUMED *END DECLARATIVES* (word)
	If the word END is encountered while processing a DECLARATIVES procedure and the word following is not DECLARATIVES or PROGRAM, the compiler assumes it to be END DECLARATIVES.

Type	Message
S	*END PROGRAM* DETECTED WITHIN DECLARATIVES SECTION (word) The END DECLARATIVES statement was not found while processing the DECLARATIVES section; however, END PROGRAM was encountered. Thus, the entire PROCEDURE division was probably processed as being contained within the DECLARATIVE section.
S	*ENTER* ROUTINE-NAMES CANNOT BE QUALIFIED (word)
S	*ERROR* REQUIRED IN ON SIZE ERROR OPTION (word)
W	EXAMINE IDENTIFIER IS ALPHABETIC AND AT LEAST ONE LITERAL IS NOT ALPHABETIC.
S	EXAMINE IDENTIFIER IS NOT USAGE DISPLAY (word)
W	EXAMINE IDENTIFIER IS NUMERIC AND AT LEAST ONE LITERAL IS NOT NUMERIC.
S	EXIT VERB MUST COMPRISE A PARAGRAPH BY ITSELF.
S	FILE-NAME CANNOT REDEFINE ANOTHER DATA-NAME.
S	FILE-NAME IS NOT UNIQUE (word)
S	FILE-NAME IS UNDEFINED (word)
S	FILE-NAME MAY NOT BE AN OCCURRING ITEM.
W	*FINIS* DETECTED BEFORE *END PROGRAM*--ASSUMED END PROGRAM (word)
W	*FINIS* OR *IDENTIFICATION* DO NOT FOLLOW *END PROGRAM*--ASSUME STACKED JOB (word) The compiler assumes the job does not contain stacked compilations.
S	FIRST ELEMENT IS INVALID IN THE MOVE STATEMENT (word) The first element must be a data-name, literal, SYSTEM-DATE, SYSTEM-TIME, CORR, or CORRESPONDING. It does not meet the specifications of a programmer-assigned word.
S	FIRST ITEM COPIED WAS NOT A PROCEDURE NAME.
S	FIRST OPERAND IN THE CONDITIONAL STATEMENT IS INVALID (word) The first operand does not meet the specification of a programmer-assigned word. It must be a data-name, literal, NOT, unary minus, or a left parenthesis.
S	*FROM, BY OR UNTIL* EXPECTED IN PERFORM VARYING STATEMENT (word) The PERFORM VARYING option syntax must be exact. FROM, BY, or UNTIL is expected and has not been found.
S	*FROM* IS MISSING IN SUBTRACT CORRESPONDING STATEMENT (word) The reserved word FROM is required at this point in the CORRESPONDING option of the SUBTRACT statement.
S	*GIVING* OPTION REQUIRED WITH *BY* AND *INTO FOLLOWED-BY-LITERAL* OPTIONS OF DIVIDE (word) If BY is used, or INTO with a non data-name operand is used, the GIVING option is required in the DIVIDE statement.
S	HIGHEST LEVEL QUALIFIER IS NOT UNIQUE (word)
S	IDENTIFIER DESCRIBING THE TABLE IN SEARCH STATEMENT MUST CONTAIN AN OCCURS CLAUSE (D idn)

Type	Message
S	IDENTIFIER DESCRIBING THE TABLE IN SEARCH STATEMENT MUST CONTAIN INDEXED BY CLAUSE (D idn)
S	IDENTIFIER ENCOUNTERED NOT FOUND IN REPORT SECTION/DATA DIVISION OR WAS NOT PROPER TYPE (word)
S	IDENTIFIER MUST REPRESENT A REPORT GROUP NAMED IN THE REPORT SECTION (word)
S	IDENTIFIER SPECIFIED IN TIMES OPTION IS NOT NUMERIC (word)
S	IF ONE PARAGRAPH-NAME APPEARS IN A SECTION, ALL PARAGRAPH-NAMES MUST APPEAR IN SECTIONS (word)
S	IF *GO TO-PERIOD-* IS SPECIFIED IT MUST APPEAR IN SENTENCE BY ITSELF (word)
S	IF *WITH* IS SPECIFIED IN OPEN STATEMENT, *NO REWIND* MUST FOLLOW (word)
S	ILLEGAL ATTEMPT TO PERFORM A SECTION WITHIN A SORT PROCEDURE. The remainder of the PROCEDURE DIVISION must not contain any transfers of control to points inside the SORT input and output procedures.
S	ILLEGAL CHARACTER IN ARITHMETIC OPERATOR (word)
S	ILLEGAL DATA-NAME (word) The current operand (word) does not meet the requirements of a programmer-assigned word; it should be a data-name.
S	ILLEGAL FILE-NAME (word) The current operand (word) does not meet the specifications of a programmer-assigned word; it should be a file-name.
S	ILLEGAL FIRST ELEMENT OF ADD/SUBTRACT STATEMENT (word) The first element must be a data-name, floating-point literal, numeric literal, integer literal, CORR, or CORRESPONDING.
S	ILLEGAL MNEMONIC-NAME (word) The current operand (word) does not meet the requirements of a programmer-assigned word; it should be a mnemonic-name.
S	ILLEGAL OPERAND FOR INITIATE, GENERATE, TERMINATE STATEMENT (word) The current operand (word) does not meet the specifications of a programmer-assigned word. For a GENERATE statement it should name a TYPE DETAIL report group or an RD entry. For either of the other two, it should be the name of an RD entry.
S	ILLEGAL PROCEDURE-NAME (word) The current operand (word) does not meet the specifications of a programmer-assigned word; it should be a procedure-name.
S	ILLEGAL QUALIFICATION LEVEL FOUND WHILE PROCESSING *CORRESPONDING* ITEMS.
S	ILLEGAL SORT NON-SORT PROCEDURE REFERENCE.
S	ILLEGAL TO MOVE A NON-INTEGER OR FLOATING POINT NUMERIC ITEM TO A-N OR A-N EDITED (D idn)

Type	Message
S	ILLEGAL TO MOVE EDITED OR ALPHABETIC ITEM TO A NUMERIC OR NUMERIC EDITED ITEM (D idn) A numeric edited, alphanumeric edited, alphabetic, or floating-point edited data item may not be moved to a numeric edited, floating-point edited, numeric (non-integer), integer, COMPUTATIONAL-1, or floating-point item.
S	ILLEGAL TO MOVE NUMERIC OR NUMERIC EDITED ITEM TO AN ALPHABETIC FIELD (D idn) A numeric edited, floating-point edited, numeric (non-integer), integer, COMPUTATIONAL-1, or floating-point field may not be moved to an alphabetic field.
S	ILLEGAL USE OF ARITHMETIC-EXPRESSION IN CONDITIONAL STATEMENT (word) An arithmetic-expression can appear only as a subject and/or object in the relational option or as a subject in the sign option of a conditional statement.
S	ILLEGAL USE OF LITERAL IN CONDITIONAL STATEMENT (word) A nonnumeric literal can appear as the subject or object only in the relational option of a conditional statement.
S	IMPERATIVE STATEMENT MUST FOLLOW *ON SIZE ERROR, AT END, INVALID KEY* (word) Either the imperative statement is missing or an IF statement has been used; conditional statements are not legal following these options.
S	IMPROPER USE OF COBOL KEYWORD (word)
S	IN NESTED IF STATEMENT *IF* WILL NOT BE EXECUTED WHEN FOLLOWS *NEXT SENTENCE* OR *GO TO*
S	IN NESTED IF STATEMENT MORE *ELSE* STATEMENTS THAN *IF* STATEMENTS WERE ENCOUNTERED.
S	IN SEARCH ALL, AND IS THE ONLY ALLOWABLE LOGICAL CONNECTOR.
S	INCOMPLETE RELATIONAL OPTION OF A CONDITIONAL STATEMENT (word) In the relational option, the first relation must be complete. The compiler has not been able to identify an object.
S	INCORRECT FORMAT IN IMPLIED OPERATOR AND CONNECTOR OPTION OF RELATIONAL CONDITION (word)
S	INDEX-DATA-ITEM ALLOWED ONLY IN SEARCH, SET, AND RELATION CONDITIONS (word)
S	INDEX-NAME ALLOWED ONLY IN PERFORM, SEARCH, SET, AND RELATION CONDITIONS (word)
S	INDEX-NAMES CANNOT BE QUALIFIED (word)
S	INITIATE, TERMINATE, AND GENERATE NOT ALLOWED IN DECLARATIVES.
S	INP MAY NOT BE USED AS MNEMONIC-NAME IN *DISPLAY*.
S	INPUT, OUTPUT, I-O, OR INPUT-OUTPUT MUST FOLLOW OPEN VERB (word) One of these reserved words is expected following the OPEN verb. The compiler cannot identify the current word.
S	*INPUT-OUTPUT* OPTION APPLICABLE ONLY FOR MASS STORAGE (word)
S	INSUFFICIENT PROCEDURE-NAMES IN GO TO DEPENDING ON STATEMENT (word) At least two procedure-names must precede the DEPENDING ON option of a GO TO statement.
S	INTEGER IN TIMES OPTION IS NOT POSITIVE (word)

Type	Message
S	*INTO* OR *BY* MUST FOLLOW FIRST OPERAND OF DIVIDE STATEMENT.
S	INVALID ADD/SUBTRACT STATEMENT (word) A data-name, numeric literal, or one of the reserved words TO, GIVING, or FROM is expected at this point in the source statement. If the word is a reserved word other than TO, GIVING, or FROM, the compiler considers itself lost and proceeds to the next keyword.
S	INVALID ALTER STATEMENT (word) The first TO of the TO PROCEED TO phrase is required; if PROCEED is specified, the second TO must follow.
S	INVALID ARITHMETIC EXPRESSION (word) An arithmetic operator and the left parenthesis must be followed by a variable, a unary minus, or a left parenthesis. A unary minus must be followed by a variable or a left parenthesis.
S	INVALID *AT END* OF SEARCH STATEMENT (word) The AT END directive is optional, but if AT is specified, END must be present.
S	INVALID CLOSE STATEMENT (word) When the word WITH is specified, the NO REWIND or LOCK words must be given. If the word NO is specified, the word REWIND must follow.
S	INVALID COMPUTE STATEMENT (word) The expression following the equal sign must be a numeric data-item, literal, or arithmetic expression. The current operand (word) is not a unary minus, left parenthesis, or a programmer-assigned word.
S	INVALID ELEMENT WITHIN DISPLAY STATEMENT (word) Data-names, literals, or figurative constants other than ALL are expected. The current word does not meet the specifications of a programmer-assigned word.
S	*INVALID KEY* OPTION OF READ STATEMENT VALID ONLY FOR RANDOM ACCESS MASS STORAGE FILES (word)
S	*INVALID KEY* OF *WRITE* AND *I/O* OF *OPEN* OPTIONS APPLICABLE ONLY FOR MASS STORAGE (word)
S	INVALID OPERAND IN SORT STATEMENT (word) The current word is not identifiable. The compiler was looking for another data-name or one of the following reserved words: ON, ASCENDING, DESCENDING, INPUT, or USING.
S	INVALID USE OF FIGURATIVE CONSTANT *ALL* (word) The literal following all must be either a nonnumeric literal or a figurative constant other than ALL.
S	INVALID WRITE ADVANCING STATEMENT (word) In the advancing option of the WRITE statement, only data-names, mnemonic-names, and integer literals are allowed. The current word does not meet the specifications of a programmer-assigned word.
S	LANGUAGE-NAME OF ENTER STATEMENT MUST BE COMPASS, FORTRAN, OR COBOL (word)
S	LITERAL OTHER THAN FIG CON *ALL* EXPECTED (word)
S	MASS STORAGE WRITE STATEMENT MUST SPECIFY INVALID KEY (word)
S	MAXIMUM OF 48 LEVELS OF KEY DATA-NAMES MAY BE TESTED IN THE SEARCH ALL CONDITION (D ıdn)

Type	Message
S	MINIMUM OF TWO OPERANDS MUST PRECEDE GIVING OPTION OF ADD.
W	MNEMONIC-NAME GIVEN IS NOT UNIQUE--*CONSOLE* USED INSTEAD.
S	MNEMONIC-NAME NOT DEFINED AS 1-CHAR LIT IN SPECIAL-NAMES (word)
S	MNEMONIC-NAME USED IS EQUATED TO AN ILLEGAL DEVICE FOR USE IN *ACCEPT* (OUT OR PUN) (word)
S	MULTIPLY **** BY NUMERIC LITERAL REQUIRES *GIVING* OPTION (word)
W	NAME USED IS NOT A MNEMONIC-NAME--*CONSOLE* USED INSTEAD.
S	*NEXT SENTENCE* OR A VERB EXPECTED IN CONDITIONAL STATEMENT (word)
	The words NEXT SENTENCE or a verb is expected at this point in the conditional statement.
S	NO SUBJECT DATA-NAME WAS ENCOUNTERED IN *SET* STATEMENT-(D idn)
S	NON-EXECUTABLE STATEMENTS FOLLOW STOP RUN OR UNCONDITIONAL GO TO (word)
	The compiler has encountered statements following STOP RUN or an unconditional GO TO within the same paragraph. Those statements can never be executed.
S	NON-NUMERIC ITEM CANNOT BE USED IN *SET* STATEMENT-(D idn)
S	NON SORT FILE ENCOUNTERED WHERE SORT FILE REQUIRED (word)
S	NON-UNIQUE MNEMONIC-NAME USED IN *WRITE---ADVANCING* (word)
S	NON-UNIQUE PROCEDURE-NAME REFERENCED
S	NON-UNIQUE SUBSCRIPT IS NOT QUALIFIED (word)
S	*NOT* OPTION CANNOT APPEAR WITHIN CONDITION PART OF A *NOT CONDITION*.
W	*NOTE* STATEMENT FOLLOWING A PARAGRAPH-NAME IS TERMINATED BY NEXT PARAGRAPH-NAME (word)
	When the NOTE statement appears as the first sentence of a paragraph, the entire paragraph is considered to be part of the NOTE character string. The compiler assumes the current element in the A-margin to be a procedure-name. When a NOTE sentence appears as other than the first sentence of a paragraph, the commentary ends with the first instance of a period followed by a space.
S	NUMERIC ITEM IN *SET* STATEMENT MUST BE LESS THAN 18 NUMERIC DIGITS-(D idn)
S	NUMERIC OPERAND IS GREATER THAN 18 DECIMAL DIGITS-(D idn)
	An attempt has been made to use the identifier (D idn) as an operand in an arithmetic statement. The size of such operands may not exceed 18 digits. The word indicated in the message is the internal name assigned by the compiler to the item. Reference the DATA DIVISION source listing to determine the actual name of the item involved.
S	OBJECT OF A SET VERB MUST NOT BE A LITERAL.

Type	Message
S	ONE CHARACTER LITERAL OR FIGURATIVE CONSTANT-EXCEPT ALL--EXPECTED AT THIS POINT IN EXAMINE (word)
S	ONLY INTEGERS ARE ALLOWED WITH TRACE CONTROL OPTIONS (word)
S	ONLY ONE *WHEN* OPTION ALLOWED IN SEARCH ALL (word)
S	OPEN, READ, SEEK, AND WRITE ARE NOT ALLOWED IN DECLARATIVES SECTION.
S	ONLY USE BEFORE/AFTER BEGINNING ARE VALID LABEL PROCEDURES ON MASS STORAGE (word)
S	OPERAND FOLLOWING *BEFORE/AFTER* OR *BEFORE/AFTER STANDARD* IS NOT IDENTIFIABLE.
W	OPERAND OF PREVIOUS DIAGNOSTIC ASSUMED CORRECT IN ORDER TO CONTINUE SYNTAX CHECK (word) The compiler assumes the operand to be as stated in the previous diagnostic in order to enable continuation of syntax checking within the current statement. This message does not clear the previous error; it indicates that the remainder of the statement is checked for additional syntax errors.
W	PARAGRAPH HEADER PRECEDING THE COPY STATEMENT IS EXPECTED AT THIS POINT (word)
S	PARAGRAPH-NAME AND PROCEDURAL PARAGRAPH MUST FOLLOW USE STATEMENT (word)
S	PARAGRAPH-NAME ASSIGNED TO A *NOTE* STATEMENT MAY NOT BE REFERENCED (word) When the NOTE verb immediately follows a paragraph-name, the name is a component of the NOTE statement and thus cannot be referenced by any procedural statements.
S	PARAGRAPH-NAME EXPECTED TO FOLLOW SECTION-NAME (word)
S	PARAGRAPH-NAME NOT UNIQUE WITHIN ITS SECTION (word)
S	PARAGRAPH-NAME PREVIOUSLY USED AS A SECTION NAME (word)
W	PREVIOUS DIAGNOSTIC SUSPENDED SYNTAX CHECK--RESUMED AT THIS POINT (word)
S	PROCEDURAL SEQUENCE CONTROL IDENTIFIER MAY NOT BE FLOATING POINT-(D idn) The control identifier for GO TO DEPENDING ON, PERFORM, and WRITE statements may not be floating-point items.
S	PROCEDURAL SEQUENCE CONTROL IDENTIFIER MUST BE AN INTEGER-(D idn) The procedural sequence controlling identifier-(D idn) for GO TO DEPENDING ON, PERFORM, and WRITE statements must be defined as having no positions to the right of its assumed decimal point. This numeric item is not an integer.
S	PROCEDURAL SEQUENCE CONTROL IDENTIFIER MUST BE NUMERIC ITEM-(D idn) The procedural sequence controlling identifier (D idn) for GO TO DEPENDING ON, PERFORM, and WRITE statements must be defined as a numeric elementary item with no positions to the right of its assumed decimal point. The identifier (D idn) is not numeric.

Type	Message
S	PROCEDURAL STATEMENT EXPECTED TO FOLLOW A PARAGRAPH-NAME (word) A procedure statement verb was not detected following a paragraph-name.
S	PROCEDURAL VERB EXPECTED BUT WAS NOT FOUND (word)
S	PROCEDURE DIVISION HEADING MISSING OR ILLEGAL.
S	*PROCEDURE* MUST FOLLOW *ERROR* OR *LABEL* IN DECLARATIVES (word)
S	*PROCEDURE* MUST FOLLOW *INPUT* AND *OUTPUT* OF SORT STATEMENT (word)
W	PROCEDURE-NAME EXPECTED IN *A* MARGIN (word) The current word does not meet the specifications of a programmer-assigned word; a procedure-name was expected.
S	PROCEDURE-NAME COPIED IS NOT EQUAL TO THE PROCEDURE NAME ON THE SOURCE STATEMENT.
S	PROCEDURE-NAME OR DECLARATIVES HEADER EXPECTED (word) A procedure-name or the DECLARATIVES header must follow the PROCEDURE DIVISION header. They must start in the A-margin of the source card.
W	PROCEDURE-NAMES AND PARAGRAPHS MUST HAVE TERMINAL PERIOD.
S	QUALIFICATION OF PROCEDURE NAMES NOT ALLOWED IN A SORT STATEMENT.
S	RECORD-NAME MUST FOLLOW RELEASE VERB (word) The current operand does not meet the specifications of a programmer-assigned word. It should be a record-name.
S	*RECORD* REQUIRED AT THIS POINT IN SYNTAX.
W	*REEL* AND *LOCK* OPTIONS APPLICABLE ONLY FOR MAGNETIC TAPE (word)
S	REEL OPTION NOT APPLICABLE TO MASS STORAGE FILES (word)
S	*RELEASE* FROM IDENTIFIER MUST BE IN INPUT RECORD AREA OR WORKING STORAGE SECTION (word)
S	*RELEASE* RECORD-NAME AND IDENTIFIER CAN NOT BE SAME DATA ITEM (word)
S	*RELEASE* RECORD-NAME MUST BE ASSOCIATED WITH A SORT FILE (word)
S	RELEASE STATEMENT APPEARS OUTSIDE A SORT INPUT PROCEDURE--NOTHING GENERATED.
S	*REPLACING/TALLYING* MUST FOLLOW EXAMINE DATA-NAME AND PRECEDE *UNTIL/FIRST/ALL/ LEADING*. Either REPLACING or TALLYING must be specified following the data-name. The next element specified must either be UNTIL FIRST (UNTIL is optional if following REPLACING), ALL, or LEADING.
S	REPORT-NAME USED WHERE A FILE-NAME WAS EXPECTED.
W	RESERVED WORD ENCOUNTERED OUT OF CONTEXT (word)
S	*RETURN* INTO IDENTIFIER MUST BE IN OUTPUT RECORD AREA OR WORKING-STORAGE SECTION (word)
S	RETURN STATEMENT APPEARS OUTSIDE A SORT OUTPUT PROCEDURE--NOTHING GENERATED.

Type	Message
S	*REVERSED* AND *REWIND* OPTIONS APPLICABLE ONLY FOR MAGNETIC TAPE (word)
W	*REVERSED* OPTION LEGAL ONLY FOR SINGLE REEL FILE (word)
S	*RUN* OR A LITERAL OTHER THAN *ALL* MUST FOLLOW THE STOP VERB (word)
S	SEARCH ALL CONDITIONS MUST BE RELATION WITH EQUAL OPTIONS OR CONDITION NAME.
S	SEARCH ALL KEY DATA-NAMES MUST BE TESTED INCLUSIVELY- HIGHEST TESTED THROUGH LOWEST.
S	SEARCH VARYING IDENTIFIERS DESCRIPTION MUST IMPLY INTEGER OR CONTAIN USAGE IS INDEX (D idn)
S	SECTION-NAME NOT UNIQUE (word)
S	*SEEK*, *INVALID KEY* OF READ, VALID ONLY FOR RANDOM ACCESS MASS STORAGE FILES (word)
S	*SENTENCE* MUST FOLLOW *NEXT* IN A CONDITIONAL STATEMENT--ASSUMED *SENTENCE* (word)
	The compiler assumes NEXT SENTENCE in order to continue syntax checking.
S	SIGN IS ILLEGAL WITH AN INTEGER SUBSCRIPT.
S	SIZE OF ENTER ROUTINE NAMES CANNOT EXCEED 8 CHARACTERS (word)
S	SORT-FILE-NAME USED WHERE A FILE-NAME WAS EXPECTED.
S	SORT INPUT PROCEDURE DOES NOT CONTAIN A RELEASE STATEMENT.
S	SORT INPUT PROCEDURE IS NOT A SECTION NAME.
S	SORT OUTPUT PROCEDURE DOES NOT CONTAIN A RETURN STATEMENT
S	SORT OUTPUT PROCEDURE IS NOT A SECTION NAME.
S	SORT PROCEDURES OVERLAP
	SORT input and output procedures must consist of one or more sections that are written consecutively and do not form a part of one another.
S	SORT STATEMENT IS ILLEGAL IN DECLARATIVES OR SORT PROCEDURE.
W	STATEMENT WHICH CAUSED PREVIOUS DIAGNOSTIC APPEARS INCOMPLETE (word)
	According to the status of the compiler when the previous diagnostic was generated, the statement in error appeared to be incomplete in addition to containing the error condition stated in the diagnostic.
S	SUBJECT AND OBJECT OF RELATIONAL CONDITION ARE BOTH LITERALS.
S	SUBJECT OF A CLASS TEST MUST BE USAGE DISPLAY--IMPLICITLY OR EXPLICITLY--(word)
S	SUBJECT OF A CONDITION-NAME CONDITION MUST BE A CONDITION-NAME (word)
S	SUBJECT OF A NUMERIC CLASS TEST CANNOT BE ALPHABETIC (word)
S	SUBJECT OF A *SIGN* CONDITION MUST BE NUMERIC (word)
S	SUBJECT OF AN ALPHABETIC CLASS TEST CANNOT BE NUMERIC (word)
S	SUBJECT OF EACH CONDITION MUST BE IN KEY IS CLAUSE OF SEARCH ALL IDENTIFIER (D idn)
S	SUBJECT OF *SET* VERB IS ILLEGAL ACCORDING TO THE OBJECT(S) ENCOUNTERED-(D idn)
S	SUBSCRIPT CANNOT BE AN OCCURRING ITEM (word)

Type	Message
S	SUBTRACT ****FROM *LITERAL* REQUIRES GIVING OPTION (word)
W	TERMINAL PERIOD NOT FOUND WHERE EXPECTED IN PROCEDURAL STATEMENT (word)
S	THE CURRENT NAME MUST BE A RECORD DESCRIPTION ENTRY (word)
S	THE FIGURATIVE CONSTANT *ALL* IS NOT ALLOWED IN TRACE (word)
S	THE SEARCH IDENTIFIER MUST NOT BE SUBSCRIPTED OR INDEXED (word)
S	*TO* MISSING IN A GO TO STATEMENT (word)
	The word TO may not be omitted. The compiler assumes (word) is a procedure-name in order to continue syntax checking.
S	*TO* MUST FOLLOW *EQUAL* IN RELATIONAL CONDITIONAL (word)
S	*TO* MUST FOLLOW FIRST OPERAND OF MOVE STATEMENT (word)
S	*TO* REQUIRED IN ADD CORRESPONDING (word)
S	UNARY MINUS MUST BE FOLLOWED BY A LEFT PARENTHESIS, IDENTIFIER OR NUMERIC LITERAL (word)
S	UNBALANCED PARENTHESES ENCOUNTERED IN A CONDITION.
S	UNDEFINED DATA-NAME (word)
W	UNDEFINED MNEMONIC-NAME--*CONSOLE* USED INSTEAD.
S	UNDEFINED NAME USED AFTER *WRITE--ADVANCING* (word)
S	UNDEFINED SUBSCRIPT (word)
S	UNIDENTIFIABLE WORD FOLLOWING CONDITION PART OF *PERFORM* (word)
W	*UNIT* OPTION APPLICABLE ONLY FOR SEQUENTIAL MASS STORAGE (word)
S	UNRECOGNIZABLE OPERAND IN A CONDITIONAL STATEMENT (word)
	A variable (i.e., data-name or literal) has been encountered or is assumed if a severe diagnostic appears above but the next operand cannot be identified. The operand must be an arithmetic operator, condition-name, NOT, relational operator, NUMERIC, ALPHABETIC, POSITIVE, NEGATIVE, ZERO, or right parenthesis.
S	*UNTIL* OPTION MAY APPEAR ONLY 3 TIMES IN PERFORM STATEMENT (word)
S	USE LABEL PROCEDURES ARE NOT APPLICABLE TO SCRATCH FILES (word)
S	*USE* STATEMENT MUST FOLLOW EACH SECTION-NAME IN DECLARATIVE SECTION (word)
S	WHEN A DATA-NAME IS INDEXED THE OPERATOR + OR - MUST BE FOLLOWED BY INTEGER.
S	*WHEN CONDITION-1* REQUIRED IN SEARCH STATEMENT (word)
S	*WHEN* IS VALID ONLY WITHIN A SEARCH STATEMENT.
S	WORD INAPPROPRIATE IN *COPY REPLACING* STATEMENT (word)

GENERATOR

Type	Message
W	A/N EDIT OF ITEM GREATER THAN 4095 CHAR--EXCESS CHAR TRUNCATED ON RIGHT.
W	BCD TOO LARGE FOR BINARY FIELD--ZERO IS MOVED.
S	DUPLICATE PROCEDURE-NAME WITHIN A SECTION.
W	FLOATING POINT-BCD NUMERIC COMPARISON MAY FAIL.
W	FRACTIONAL PORTION TRUNCATION OCCURS ON BCD-BINARY CONVERSION.
S	IDENTIFIER IN SET UP BY OR DOWN BY MAY NOT BE A FLOATING POINT ITEM.
S	IDENTIFIER IN SET UP BY OR DOWN BY MAY NOT BE AN INDEX DATA ITEM.
S	ILLEGAL DECLARATIVES PROCEDURE REFERENCE.
S	ILLEGAL TO PERFORM PROCEDURE IN DECLARATIVES THRU PROCEDURE IN NON-DECLARATIVES.
S	INVALID PERFORM--PROCEDURE-NAME-2 OCCURS BEFORE PROCEDURE-NAME-1.
S	LEFT TRUNCATION MAY OCCUR ON BCD-BINARY CONVERSION.
W	LOSS OF SIGNIFICANCE MAY OCCUR ON NUMERIC EDIT.
S	NUMBER OF SUBSCRIPTS GIVEN DOES NOT EQUAL NUMBER REQUIRED.
S	PROCEDURE-NAME UNDEFINED WITHIN SECTION QUALIFIER.
W	TALLY IS 5 DIGITS AND MAY OVERFLOW ON EXAMINE.
S	UNDEFINED PROCEDURE NAME (word)

GENERAL COBOL RULE VIOLATION DIAGNOSTICS

Type	Message
W	CHARACTER IN CONTINUATION FIELD NOT A HYPHEN--ACCEPTED AS HYPHEN.
S	COMPILER CANNOT READ THE LIBRARY DIRECTORY FOR COPY.
S	COMPILER CANNOT READ THE LIBRARY FOR COPY.
S	COPY IS NOT ALLOWED WITHIN A RENAMED FILE OR WITHIN TEXT COPIED FROM THE LIBRARY.
E	FIRST NON-BLANK CHARACTER IN CONTINUED NON-NUMERIC LITERAL NOT A QUOTE--ACCEPTED AS QUOTE.
S	LIBRARY-NAME IN COPY STATEMENT CANNOT BE FOUND IN LIBRARY DIRECTORY.
E	ITEM USES MORE THAN 3 SUBSCRIPTS
E	ITEM USES MORE THAN 49 LEVELS OF QUALIFICATION
S	LIBRARY-NAME IN COPY STATEMENT MAY NOT BE COBOL RESERVED WORD AND MUST BE 8 CHARS OR LESS.
S	NON-COBOL CHARACTER IN SOURCE TEXT--THE FOLLOWING CHARACTER IS ILLEGAL (word)
E	NON-NUMERIC LITERAL EXCEEDS 120 CHARACTERS--FIRST 120 ACCEPTED.
S	NON-USASI ELEMENT-CLAUSE IGNORED.
	This diagnostic is produced only when an assembly option is exercised by the installation. This error is always classed as severe.
	TERMINAL PERIOD MISSING (word)
E	WORD EXCEEDS 30 CHARACTERS--FIRST 30 ACCEPTED.

FATAL ERROR CONDITIONS

A system error which prohibits compilation from continuing causes a message of the following format to be written on the job standard out file.

Message	Significance		
*UCBL x=n function REJECT ON DSI dsi. Action phrase.	x	x or y returned by blocker/deblocker	
		x for LOCATE or SEXPAND	MASTER REFERENCE MANUAL
	n	Corresponding reject code.	
	Function	Any one of the following: PACK, PACKC, PACKR, PICK, PICKC, PICKD, LOCATE, SEXPAND, ASSIGNM, SALOCATE	
	Action phrase	Optional description applicable to error message.	

OBJECT-TIME DIAGNOSTICS

REPORT WRITER

During program execution the routines generated to produce a report test for four invalid conditions. The following corresponding messages are written on the job standard OUT file. The job is then terminated with UCBL ABNORMAL TERMINATION written on the standard OUT file.

```
report-name INITIATE ON INITIATED REPORT
report-name TERMINATE ON UN-INITIATED REPORT
report-name GENERATE ON UN-INITIATED REPORT
report-name REACHED END OF ALLOCATED FILE
```

USASI COBOL/SORT

At the completion of a USASI COBOL/SORT operation, the following informative message is written on the standard OUT file.

```
UCBL SORT RECORDS IN xxxxxxxx RECORDS OUT yyyyyyyy
```

At the beginning of the sort operation, if insufficient core is scheduled to allow sorting to begin, the following voluntary abort message is generated for the job.

```
UCBL SORT INSUFFICIENT CORE SCHEDULED
```

The user should schedule more core before attempting to run the job again.

At the end of a SORT operation, if the record counts do not agree the following message is displayed to the operator.

```
UCBL SORT RECORD COUNTS DO NOT AGREE ACCEPT ABANDON
```

Operator action is as follows:

1. Press MANUAL INTERRUPT
2. Type response code Rr,
3. Type either ACCEPT or ABANDON
4. Press FINISH

RERUN/RESTART MESSAGES

Message	Significance
RERUN DUMP number of dump just written	Written on job standard OUT file as each rerun dump is taken during program execution.
RERUN FILE FULL. LAST DUMP # IS number of last complete dump written	Subsequent dump requests are ignored. Dump file may become full if allocated area for mass storage file is exceeded, or magnetic tape file end-of-reel is reached. Either expand mass storage dump file or mount new output tape reel. Restart from any dump up to and including last complete dump.

SYSTEM REJECT CONDITIONS

A job may abort because of conditions detected by the MASTER BLOCKER/DEBLOCKER, MIOCS, *DEF, or system OCARE processors. The format of such diagnostics is as follows:

Message	Significance
UCBL I/O ERR s mmcc DSI dsi	s Code indicating the object-time processor rejected by MASTER. O Open request processor C Close request processor R Read request processor W Write request processor M Move processor H Hardware failure processor S Seek request processor D Rerun/restart processor E Multifile processor U Unknown reject processor mm 2-digit code indicating which MASTER routine caused reject. cc 2-digit error code returned by MASTER routine on rejection. mm Routine 01 BLOCKER/DEBLOCKER 02 MIOCS 03 *DEF 04 System OCARE

TERMINATIVE DIAGNOSTICS

When the object-time I/O system encounters a trouble area for which no correction can be made, the system generates an abortive diagnostic on the standard OUT file for the job. The format of the message is indicated below; the error code depends on the source code for interpretation.

UCBL I/O ERROR s nnnn DSI dsi

s	Code indicating which I/O routine generates the diagnostic.
nnnn	Numeric code for the error encountered.

s	nnnn	Significance
C	0001	Attempt to CLOSE an unopened file.
D	0001	Abnormal termination of a read/write operation in the rerun dump routine.
H	0001	Hardware device is inaccessible; processing cannot continue.
O	0001	Attempt to OPEN WITH NO REWIND on a file not previously CLOSED WITH NO REWIND.
R	0001	Attempt to READ a file which was not opened for INPUT or INPUT-OUTPUT.
S	0001	User requested a SEEK on an unopened file.
W	0001	Attempt to WRITE on a file declared to be INPUT only.
C	0002	Attempt to CLOSE REEL on a mass storage file.
D	0002	Rerun file dsi error in the restart control card; control card contains more than four characters.
H	0002	Operator decided to abandon the job after encountering an irrecoverable I/O transmission error where the user provided no AFTER STANDARD ERROR RECOVERY declarative procedure.
M	0002	Logical record size error. A logical record exceeds the maximum size defined for the file.
O	0002	Attempt to OPEN WITH NO REWIND a mass storage file.
R	0002	Attempt to READ an unopened file.
S	0002	User attempted a SEEK on a sequential access file.
W	0002	Attempt to WRITE on an unopened file.
D	0003	Abnormal termination of a LOCATE, PICK, or READ request in the restart routine.
M	0003	Variable portion of a logical record exceeds the maximum variable portion size defined for the file.
O	0003	Attempt to OPEN a file which is already open.
R	0003	Attempt to READ a file which has reached EOF but has not been closed.
W	0003	Attempt to WRITE on a magnetic tape file which has been CLOSED WITH NO REWIND or CLOSED WITH REWIND but has not been subsequently opened again.
O	0004	Attempt to open a file that shares areas with or is on the same multifile reel as another file which is currently open.
W	0004	Attempt to WRITE on a mass storage file following a previous WRITE which was rejected; control had been passed to the user's INVALID KEY procedure.
O	0005	Abnormal termination of a forward I/O search on a multifile reel.

s	nnnn	Significance
O	0006	Abnormal termination of a tape rewind.
W	0006	User program attempted two successive WRITE requests without an intervening READ request on an INPUT-OUTPUT file.
O	0007	Attempt to OPEN INPUT REVERSED a file with variable length blocked records.
E	00yy	End-of-reel condition exists on an OUTPUT multifile reel. yy is the position number of the file being created at the time the error occurred.
R	0106	A PICK request was rejected when the blocker/deblocker routine tried unsuccessfully to mount a new segment on the file.
R	0107	Defined block area out of bounds to read routine.
U	02xx	Abnormal status reject on an MIOCS function request; xx is the error code returned by MIOCS (section 2.1, MASTER Diagnostic Handbook, Pub. No. 60206800.
L	010x	Reject on a BLOCKER/DEBLOCKER request when attempting to PICK a STANDARD label card from a file on SYSTEM-INPUT when file is declared to contain labels. x is error code returned by blocker/deblocker (section 2.3, MASTER Diagnostic Handbook, Pub. No. 60206800.

OPERATOR MESSAGES

MESSAGE TO OPERATOR						
Type	Job	Task	nnn	Optional Message	Significance	Action
Rr	JOB i	UCBL		I/O IRR ERR ON DSI dsi ACCEPT BYPASS RETRY ABANDON	Irrecoverable error on I/O. User program did not specify AFTER STANDARD ERROR RECOVERY DECLARATIVE. Operator must make appropriate decision.	1. Press MANUAL INTERRUPT 2. Type MASTER generated response code and one action word: Action word ACCEPT Ignore error condition and proceed. BYPASS Skip erroneous record and proceed. RETRY Repeat recovery procedures. ABANDON Terminate job. 3. Press FINISH
Rr	JOB i	UCBL		LBL ERROR DSI dsi REEL nn ht CcEeUuuu	Header label does not agree with information furnished by user.	1. After mounting correct reel, type Rr, OK 2. To ignore incompatibility, type Rr, NO 3. Press FINISH
Rr	JOB i	UCBL		MNT INP/OUT DSI dsi REEL nn ON ht CeEeUuuu RSVP		1. Mount reel n on tape unit uu 2. Type response: Rr, OK Reel mounted Rr, NO Request cannot be honored 3. Press FINISH
A	JOB i	UCBL		MNT INP/OUT DSI dsi REEL nn ON ht CcEeUuuu	File is assigned to alternating units. Object-time I/O system automatically switches to alternate reel at reel end.	Mount reel nn on unit uuu. No response required.
A	JOB i	UCBL		MNT INP/OUT DSI dsi REEL nn ON UNIT NEXT ASSIGNED TO THIS JOB		Mount reel nn on unit assigned by the *DEF logging message immediately following this message.

ENTER VERB OBJECT CODE EXAMPLES J

The ENTER verb object code varies when data-names are all alphanumeric, all numeric, or mixed. The following symbols are used in the VFD (variable field definition) in the examples:

Characters preceding slash marks (mode indicators)

O	Octal
A	Word address arithmetic
C	Character address arithmetic

The positive decimal integer following the mode indicator denotes the number of bit positions in the variable field.

Characters following slash marks

L	location
S	length
0	zero
P	point location

Characters in parentheses denote the data-name, file-name or procedure name associated with the indicator. For example, S(DN3) means the size of DN3.

ENTER USASI COBOL Subprograms

ENTER COBOL; XCBLSUB.

Resulting object code:

```
EXT         XCBLSUB
RIS
RTJ         XCBLSUB
ROS
```

Parameters may not be passed to a COBOL subprogram. Instead, the two programs communicate through the Common-Storage section of the DATA Division.

ENTER COMPASS/MASTER Subprograms

ENTER COMPASS; SUBR, DN1, DN2, DN3, FN1, PN1, C2DN, C1DN

DN	Data-name
FN	File-name
PN	Procedure-name
C2DN	COMP-2 data-name
C1DN	COMP-1 data-name

Resulting object code:

```
EXT    SUBR
RIS
RTJ    SUBR
VFD    O6/52,O1/0,C17/L(DN1)
VFD    O7/0,O17/S(DN1)
VFD    O6/60,O1/0,C17/L(DN2)
VFD    O6/P(DN3),O6/S(DN#),O6/P(DN2),O6/S(DN2)
VFD    O6/61,O1/0,C17/L(DN3)
VFD    O6/40,O3/0,A15/L(FN1)
VFD    O6/00,O3/0,A15/L(PN1)
VFD    O6/71,O3/0,A15/L(C2DN)
VFD    O6/70,O3/0,A15/L(C1DN)
ROS
```

ENTER USASI FORTRAN Subprogram

ENTER FORTRAN: SQRTFN; C2DN

C2DN	COMP-2 data-name

Resulting object code:

```
EXT    SQRTFN
RIS
RTJ    SQRTFN
VFD    O6/71,O3/0,A15/L(C2DN)
```

INDEX

CONTROL DATA
CORPORATION

COMMENT AND EVALUATION SHEET
3300/3500 USASI COBOL/MASTER
Reference Manual

Pub. No. 60229400 February 1969

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.

CUT ALONG LINE

PRINTED IN U.S.A.

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
Software Documentation
4201 North Lexington Avenue
St. Paul, Minnesota 55112

CUT ALONG LINE

MD 248

FOLD FOLD

STAPLE STAPLE

CONTROL DATA

1/2 3/4 1 1-1/4

►►CUT OUT FOR USE AS LOOSE-LEAF BINDER TITLE TAB

3300/3500 USASI COBOL/MASTER REFERENCE MANUAL

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. 60229400

Litho in U.S.A.