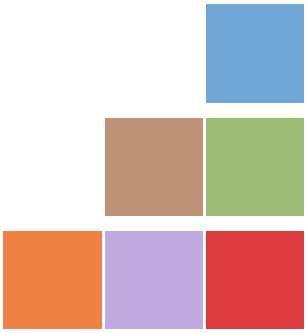
COBOL-IT® Compiler Suite Enterprise Edition

Reference Manual
Compiler version 3.10



Acknowledgment

This documentation is derived from COBOL-IT Source code, parts of which are derived from OpenCOBOL.

In 2008, COBOL-IT forked its own compiler branch, with the intention of developing a fully featured product and offering professional support to the COBOL user industry.

Conventions used in the General Format diagrams:

Brackets [ ] identify syntax elements that are supported but not required.

Curly Braces { } identify alternative syntax elements. Among syntax elements described within stacked curly braces, only one of the entries may be selected.

Ellipses (...) indicate (optional) repetition. If the syntax element is a required element, then it will be surrounded by curly braces.

Third-Party software components embedded in the SOFTWARE and Services and submitted to specific licenses:

VBISAM

* Copyright (C) 2003 Trevor van Bremen

* Copyright (C) 2008-2018 Cobol-IT

* License: LGPL

GMP (GNU Multiprecision Library)

* Copyright 1991, 1996, 1999, 2000, 2007 Free Software Foundation, Inc.

* License: LGPL

GNU LIBICONV

The libiconv libraries and their header files are under LGPL.

Contact Information:

The Lawn
22-30 Old Bath Road Newbury, Berkshire,
RG14 1QN United Kingdom
Tel: +44-0-1635-565-200

Language Reference

This Reference manual lists COBOL -IT recognized syntax.
This is not intended to be a COBOL Language reference. For formal description of each clause, please refer to general purpose COBOL literatures.

1 Source Code Management Topics

1.1 Free Source Format

Free source format, or "terminal format" applies different rules for establishing the location of the Area A, the Indicator Area, Area B, and the Identification Area, as specified below.

COBOL-IT source programs that are written in Terminal Format must be compiled with the –free compiler flag.

Example:

>cobc –free hello.cbl

General Rules

1. Area A may start in column 1, 2, or 3, and extends for 4 characters. Most commonly, Area A begins in column 1.
2. The Indicator Area is optional, and can be used to mark a comment line with a "*" character. The Indicator Area must start in column 1. Note that both Area A and the Indicator Area may start in Column 1.
3. Area B starts after the last character in Area A. In the most common case, where Area A begins in column 1 and extends for 4 characters, Area B would start in column 5. Area B extends to the end of the line, or to the beginning of the Identification Area.
4. Identification Area holds in-line comments. In a program written in Terminal Format, the beginning of the Identification Area is marked with "*>" (without the quotes). Note that when contained within quotation marks, the "*>" mark is not interpreted as the beginning of the Identification Area.
5. The Identification Area extends to the end of the line.

Code Sample:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  HELLO.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 WS-DUMMY PIC X.
PROCEDURE DIVISION.
DISPLAY "HELLO WORLD" LINE 10 COL 10.
ACCEPT WS-DUMMY.
STOP RUN.
```

6. Note that COBOL-IT's default behavior is to assume an ANSI format, and the compiler will flag a terminal format source file with an error such as the following:

7. C:\COBOL\CobolIT>cobc hello.cbl

8. C:/COBOL/CobolIT/hello.cbl:1: Error: Invalid indicator 'f' at column 7

9. To correct this error, compile with the –free compiler flag:

10. C:\COBOL\CobolIT>cobc -free hello.cbl
    cob44D4.c
    Creating library hello.lib and object hello.exp

1.2 Line Continuations

A line continuation condition exists when a clause continues in Area B of the next line (excluding comment lines).

General Rules

1. A continuation line is marked by placing a hyphen "-" in the indicator area.
2. On a continuation line-

   If the continued line ends in a literal without a closing quotation mark, then the first non-blank character of the continuation line must be a quotation mark.

   In all other cases, the first non-blank character of the continuation line is interpreted as following the last non-blank character in the previous line.

Code Sample:

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  LINECONTINUE.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 WS-LITERAL  PIC X(75) VALUE "THIS IS A LONG LITERAL THAT IS US
      -    "EFUL IN DEMONSTRATING LINE CONTINUATIONS.".
       77 WS-DUMMY PIC X.
       PROCEDURE DIVISION.
           DISPLAY WS-LITERAL LINE 10 COL 1.
           ACCEPT WS-DUMMY.
           STOP RUN.
```

1.3 Blank Lines and Comment Lines

A blank line is a line that contains only spaces in Area A and Area B. A comment line is a line that contains an asterisk "*"in the indicator area. Blank lines and comment lines can appear anywhere in a source file. Blank lines and comment lines are preserved in source listings, but have no effect on the behavior of the compiled code.

The "*>" notation may be used anywhere in a source line to designate a comment and both single-quote, and double-quote notations may be used in the comment.

1.4 Conditional Compilation

1.4.1 $END Statement

The $END Statement terminates the conditional compile construct.

General Format

```
$END
```

General rules

The $END statement must appear on a single line.

1.4.2 $ELSE Statement

The $ELSE Statement indicates a branch that should be executed if the $IF condition does not test true.

General Format

```
$ELSE
```

General rules

The $ELSE statement must appear on a single line.

1.4.3 $IF Statement

The $IF Statement provides the ability to conditionally include or exclude text based on the truth test of the if-condition. There are two formats:

Format 1

```
$IF constant-name-1 [NOT] = literal-1
```

Format 2

```
$IF [X0 - X9] [NOT][=] {ON  }
                       {OFF }
```

Syntax

1. Constant-name-1 is named in the compiler flag –constant “constant-name-1=value”.
2. Literal-1 may be a numeric or alphanumeric literal.
3. The $IF/$ELSE/$END/$SET directive must not be terminated by a period.

General rules

1. If the condition evaluates "true", the source lines following the $IF statement are processed. If the condition evaluates "false", COBOL source lines are ignored until the next conditional compilation line is encountered.
2. $IF/$ELSE/$END conditional compilation directives may be nested to 99 levels.

1.4.4 $SET Statement

The Format 1 $SET Statement is used to set one of the X0-X5 switches to ON or OFF.

Format 1

```
$SET [X0-X9]={ON }
             {OFF}
```

The Format 2 $SET Statement provides a way to dynamically create a constant with a value, which may be used programmatically like a level-78 constant.

Format 2

```
$SET [constant-1] = [constant-value ]
```

Code Sample

```
. . .
$SET MYCONSTANT = 50
. . .
01 MY-BUFFER      PIC X(MYCONSTANT).
78 NUMBER-STATES  VALUE MYCONSTANT.
```

General rules

The $SET statement must appear on a single line.

Format 1 Code Sample

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  CONDCOMPILE.
* COMPILE WITH
* >COBC -CONSTANT "C1="USA"" -CONSTANT "C2=100" CONDCOMPILE.CBL
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77 WS-DUMMY     PIC X.
 PROCEDURE DIVISION.
 MAIN.
* FORMAT 1
* $IF CONSTANT-1 [NOT] = LITERAL-1


$IF C1 = "USA"
     DISPLAY "C1 = USA" LINE 6 COL 10.
```

```
      $ELSE
           DISPLAY "C1 NOT = USA" LINE 6 COL 10.
      $END

      $IF C2 = 100
           DISPLAY "C2 = 100"  LINE 8 COL 10.
      $ELSE
           DISPLAY "C2 NOT = 100"   LINE 8 COL 10.
      $END

      * FORMAT 2
      * $SET X0=ON

      $SET X0=ON
      $IF X0=ON
           DISPLAY "X0 SET!" LINE 10 COL 10.
      $ELSE
           DISPLAY "X0 NOT SET!" LINE 10 COL 10.
      $END

           ACCEPT WS-DUMMY.
           STOP RUN.
```

1.5 Use of Figurative Constants

Figurative Constants are reserved words with equivalent values.
The Figurative Constants recognized by COBOL-IT and their equivalent values are listed below:

Figurative Constant	Equivalent Value
HIGH-VALUE, HIGH-VALUES	Hex “FF”
LOW-VALUE, LOW-VALUES	Hex “00”
NULL	Hex “00”
QUOTE, QUOTES	Double-quote character (“)
SPACE, SPACES	The Space character
ZERO, ZEROS, ZEROES	The 0 character

General Rules

1. With the exception of ZERO, ZEROS, ZEROES, which are considered numeric, all of the figurative constants are considered to be alphanumeric literals.
2. Figurative constants can be used in any statement that their equivalent value can be used.

1.6 COPY Statement

The COPY statement names a file copy-lib that is to be copied into the source at the location of the COPY statement prior to compilation.

General Format

```
COPY copy-lib
  [ REPLACING  { { old-text BY new-text } } ... ]
        { { {LEADING } literal-1 BY {literal-2} } }
        { { {TRAILING}                {SPACE } } }
        { {                          {SPACES } } }.
```

Syntax

1. The COPY statement may appear in Area A or Area B.
2. Old-text requires that at least one word be specified.
3. Old-text and new-text may be any of the following:
    a. A word or series of text words placed between "= =" delimiters.
    b. A numeric or non-numeric literal
    c. A data name, including qualifiers, subscripts, and reference modification.

4. Old-text may **not** be any of the following
    a. A space
    b. The "= =" delimiter
    c. The quote character
    d. The word "COPY" designating a COPY statement

5. Literal-1 and literal-2 are nonnumeric literals.

General Rules

1. copy-lib may be entered in quotes, as in COPY "copy-lib" or not, as in COPY copy-lib. The internal rules applied for locating the copy file and resolving the copyfile name are the same. For details about rules used to locate copy books and resolve copy book names, see 6.4 COPY BOOK Handling.
2. The REPLACING clause indicates text replacements to be made prior to compiling the source.
3. When old-text is identified in copy-lib, it is replaced by new-text, at the former starting position of old-text.
4. The LEADING/TRAILING phrase indicates that only the LEADING or TRAILING characters identified will be replaced if they match text in copy-lib.
5. The SPACE or SPACES phrase indicates that old-text should be removed.
6. SPACE and SPACES are synonyms.
7. A period "." At the end of a COPY statement is not required.
8. ++COPY and ++INCLUDE are synonyms of COPY.
9. COBOL-IT recognizes the following delimiters:
    a. = = old-text = =
    b. = = : old-text : = =
    c. Note that spaces between old-text and the delimiter are ignored.

Compiler flags related to locating copy files and resolving copy file name:

-I <path>[,ext1,ext2,.,extn][@<LibName>] | <command-file>	Searches <path> for copy files, including files with extensions of ext1, ext2, ... extn, and located in directory named by <LibName>.
–ext <extension>	Includes <extension> along with default extensions
-fcopy-mark	Adds mark for begin/end of COPY In listing and preprocessed file. Note: The copy marks are:

```
*++SCOPY
[ COPY file is listed here ]
*--SCOPY
```

-fcurdir-include	Causes COPY file search to begin in current directory
-ffold-copy-lower	Fold COPY subject to lower case looking for match
-ffold-copy-upper	Fold COPY subject to upper case looking for match

Environment variables related to locating copy files:

COB_COPY_DIR=<path>	Searches <path> for copy files.
COBCPY=<path>	Searches <path> for copy files.

1.7 REPLACE Statement

The REPLACE Statement allows the user to direct the compiler to replace selected strings or numbers in source text with alternative strings or numbers.

General Format

Format 1

The Format 1 REPLACE statement indicates text replacements to be made prior to compiling the source until another Format 1 REPLACE statement is encountered, or until the REPLACE OFF statement is encountered.

```
REPLACE { { old-text BY new-text } } ... ]
        { { {LEADING } literal-1 BY {literal-2} } }
        { { {TRAILING}                 {SPACE } } }
        { {                           {SPACES } } }.
```

Syntax Rules

1. *old-text* requires that at least one word must be specified.
2. *old-text* and *new-text* may be any of the following:
   - A word or series of series of text words placed between "==" delimiters.
   - b. A numeric or nonnumeric literal.
   - c. A data name, including qualifiers, subscripts, and reference modification.
3. *old-text* may **not** be any of the following:
   - a. A space
   - b. The "= =" delimiter
   - c. The quote character
   - d. The word "COPY" designating a COPY statement.
4. *literal-1* and *literal-2* are nonnumeric literals.

General Rules

1. The REPLACE statement must be terminated by a period.
2. The REPLACE Statement may appear anywhere in the COBOL source text, in Area A or Area B.
3. Text replacement described by a REPLACE statement is done prior to compilation, after the processing of the COPY statement.
4. The REPLACE statement specifies conversion of source statements containing *old-text* into *new-text*.
5. Text replacement uses the following rules:
   a. a. The leftmost source text word is *old-text*.
   b. b. *Old-text* is matched when an identical word or sequence of words is encountered within the scope of the REPLACE statement. When matching, multiple spaces match a single space. Lower case and upper case characters match, except in quoted literals.
6. When *old-text* is matched with existing source code, it is replaced by *new-text* in the source program, in the same location.
7. When you are using the LEADING/TRAILING option, a match between *old-text* and matched source code can be made with a substring in an element in the source code.
8. When SPACE, or SPACES are indicated as *new-text*, matched *old-text* characters are deleted.

Format 2

The Format 2 REPLACE statement removes the existing REPLACE statement directive.

```
REPLACE OFF.
```

Format 3

The Format 3 REPLACE ADD nests a REPLACE statement inside an existing REPLACE statement.

```
REPLACE ADD.
```

General Rules

1. Support for the REPLACE ADD statement requires that the replace-additive compiler configuration file variable be set to on.

   The replace additive compiler configuration file variable allows for the use of the REPLACE ADD verb, which has the effect of nesting a REPLACE statement inside an existing REPLACE statement. Nested REPLACE statements are executed before outer REPLACE statements in COBOL-IT's precompile phase. Note that a a REPLACE stack can be cleared with the REPLACE OFF statement.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  REPLACE1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       REPLACE LEADING ==WS== BY ==W==.
       77 WS-DUMMY  PIC X.

       PROCEDURE DIVISION.
           REPLACE "HELLO WORLD" BY "ENJOY COBOL-IT" 10 BY 20.
           DISPLAY "HELLO WORLD" LINE 10 COL 10.
           REPLACE OFF.

           REPLACE ==LINE 10== BY ==LINE 22==
                            ==COL 10== BY ==COL 20==
                            =="HELLO WORLD"== BY =="GOODBYE FRIEND"==.
           DISPLAY "HELLO WORLD" LINE 10 COL 10.
           ACCEPT W-DUMMY.
           STOP RUN.
```

Produces the following screen output:

ENJOY COBOL-IT
GOODBYE FRIEND

1.8 Special Registers

Tally Register

The TALLY register is used during TALLYING operations.

- The predefined register TALLY is defined as:

```
01 TALLY GLOBAL PICTURE S9(9) USAGE COMP-5 VALUE ZERO.
```

Note- The configuration file flag tally-register, when set to NO, disables the creaton of the predefined register TALLY, for example: tally-register:no

Return-Code Register

The RETURN-CODE register is used by a number internal library routines, and can also be set by the STOP statement to return a value after the run unit has terminated.

- The predefined register RETURN-CODE is defined as:

```
01 RETURN-CODE  EXTERNAL   USAGE BINARY-LONG.
```

Sort-Return Register

The SORT-RETURN register is set to “0” following a successful SORT, and to a non-zero value if the SORT is unsuccessful.

- The predefined register SORT-RETURN is defined as:

```
01 SORT-RETURN USAGE BINARY-LONG.
```

Number-of-Call-Parameters Register

The NUMBER-OF-CALL-PARAMETERS stores the number of parameters with which the program was CALL’ed.

- The predefined register NUMBER-OF-CALL-PARAMETERS is defined as:

```
01 NUMBER-OF-CALL-PARAMETERS USAGE BINARY LONG.
```

2 IDENTIFICATION DIVISION.

The IDENTIFICATION Division describes the name, and type of program.

General Format

```
{ IDENTIFICATON } DIVISION.
{ ID            }
[ PROGRAM-ID. Program-n    [ IS {INITIAL   } PROGRAM ] . ]
                                {COMMON    }
                                {RECURSIVE }
```

Syntax

1. **Program-n** is a program name, which may be expressed as a data element, literal, or data returned from a function call.
2. **COMMON** may only be used in a nested program.

General Rules

3. Program-n is the program name that is recognized by the debugger, and other external tools inquiring the program name.
4. The COMMON clause indicates that the program can only be used in a nested program.
5. The INITIAL clause indicates that all file connectors and data elements are set to their initial state when the program is activated.
6. The RECURSIVE clause indicates that the program may call itself while it is activated.

3 ENVIRONMENT DIVISION.

The Environment Division describes the program's environment.

General Format

```
[ [ ENVIRONMENT DIVISION. ]
[ [ CONFIGURATION SECTION. ]
  [ SOURCE-COMPUTER. source-computer-paragraph ]
  [ OBJECT-COMPUTER. object-computer-paragraph ]
  [ SPECIAL-NAMES. [ special-names-paragraph ]
  [ REPOSITORY.    [ repository-paragraph ] ] ]
[ [ INPUT-OUTPUT SECTION. ]
  [ FILE-CONTROL. ] { file-control-entry } ...
  [ I-O-CONTROL. [ i-o-control-entry ] ] ] ]
```

Syntax

1. The division header for the Environment Division is optional.

General Rules

The General Rules for the Statements and Clauses of the Environment Division entries are described below.

3.1 CONFIGURATION SECTION

3.1.1 SOURCE-COMPUTER.

The SOURCE-COMPUTER paragraph describes the computer on which the source program is compiled.

General Format

```
SOURCE-COMPUTER.  [ src-computer-name      ].
                  [ WITH DEBUGGING MODE    ] .
```

Syntax

1. The SOURCE-COMPUTER clause is recognized, and syntax is validated. However, the SOURCE-COMPUTER clause is otherwise treated as commentary.

2. The WITH DEBUGGING MODE clause is recognized, and syntax is validated. However, the WITH DEBUGGING MODE clause is otherwise treated as commentary.

General Rules

1. COBOL-IT supports conditional debugging lines with the use of the –fdebugging-line compiler flag.

2. Note- The effect of using the –fdebugging-line compiler flag is to treat conditional debugging lines as normal source lines. Conditional debugging lines are lines that contain a “D” in column 7.

3.1.2 OBJECT-COMPUTER.

The OBJECT-COMPUTER paragraph names the computer on which the program is to be run.

General Format

```
OBJECT-COMPUTER.  [ obj-computer-name ] .
   [ MEMORY SIZE integer {WORDS     } ]
                         {CHARACTERS}
   [ PROGRAM COLLATING SEQUENCE IS alphabet-name ]
   [ SEGMENT-LIMIT IS integer-literal ] .
```

Syntax

1. obj-computer-name is a user-defined word or sequence of words that identifies the computer on which the program is run.
2. alphabet-name is an alphabet name that is described in Special-Names, or EBCDIC.
3. obj-computer-name is treated as commentary.
4. The MEMORY SIZE clause is treated as commentary.
5. The PROGRAM COLLATING SEQUENCE clause causes the program to use the collating sequence of *alphabet-name* to determine the truth value of nonnumeric comparisons.
6. The SEGMENT-LIMIT clause is recognized, and syntax is validated. However, the SEGMENT-LIMIT clause is otherwise treated as commentary.

General Rules

1. The PROGRAM COLLATING SEQUENCE clause also applies to non-numeric sort and merge keys. However, the COLLATING SEQUENCE phrase in a SORT or MERGE statement takes precedence over the PROGRAM COLLATING SEQUENCE clause.

3.1.3 SPECIAL-NAMES.

The SPECIAL-NAMES paragraph describes different aspects of the operating environment.

General Format

```
SPECIAL-NAMES.
   [ {switch-name} [ IS mnemonic-name-1 ]
     {system-name}
   [ {ON } STATUS IS switch-status ] ... ] ...
     {OFF}
   [ {alphabet-entry } ... ]
     where alphabet-entry can have format:

      ALPHABET alphabet-name IS {NATIVE}
                                {STANDARD-1}
                                {STANDARD-2}
                                {EBCDIC}
     and alphabet-entry may also have format:
                ALPHABET alphabet-name IS
              {alphabet-literal-1 [ THRU alphabet-literal-2 ] } ...
                                  [ {ALSO alphabet-literal-3  } ...]
    [ LOCALE locale-name IS locale-reference ]
```

```
[ SYMBOLIC CHARACTERS
   {   {symbol-char} … {IS }  (char-val} … } …
                       {ARE}
   [ IN alphabet-name  ]]

[ CLASS class-name IS
            { [{THROUGH} literal-2 ]}...]...
                        {THRU   }
[ CURRENCY SIGN IS currency-literal ]
[ DECIMAL-POINT IS COMMA ]
[ CALL-CONVENTION number IS mnemonic-name-1 ]
[ CURSOR IS cursor-pos ]
[ CRT STATUS IS crt-status-var ]
[ NUMERIC [ SIGN [IS] {LEADING}
                      {TRAILING}
                      {SEPARATE} ]].
```

Syntax

1. **switch-name** is one of the 36 system switches, SWITCH-1, SWITCH-2, SWITCH-3, SWITCH-4, SWITCH-5, SWITCH-6, SWITCH-7… SWITCH-36.
2. **system-name** is one of the following: CONSOLE, CSP, SYSIN,SYSIPT, SYSOUT, SYSLIST, SYSLST, SYSERR, PRINTER, TERMINAL, , .
3. **mnemonic-name-1** a user-defined word. When associated with a switch, it is used to change the state of the switch. When used with a system-name, it is used in ACCEPT and DISPLAY statements to refer to the named device.
4. **switch-status** is a user-defined word that represents the ON or OFF status of a program switch.
5. **alphabet-name** is a user-defined word.
6. **alphabet-literal-n** may be either numeric or alphanumeric. When numeric, it is restricted to the range of 1 through 256.
7. **locale-name** is a user-defined word that refers to a locale.
8. **locale-reference** is a mnemonic that can be used to refer to the locale-name.
9. **class-name** is a user-defined word that describes a class, defined by the following VALUE-1 THROUGH VALUE-2 clause.
10. **literal-n** is a numeric or alphanumeric literal.
11. **currency-literal** is a one-character alphanumeric literal used to represent a currency.
12. **cursor-pos** is the name of a numeric data item described in the working-storage section.
13. **crt-status-var** is a numeric data item described as PIC 9(4) that is declared in the working-storage section.

14. **numeric** defines a default sign storage convention, when the sign storage for a data item is not specified.
15. **symbol-char** is a user-defined word that names the symbolic-character.
16. **char-val** is an integer that indicates the ordinal position of a character in the native character set.

General Rules

The General Rules for the SPECIAL-NAMES clauses are described below.

SWITCH NAME

Switches are condition tests that are associated with switch-statuses. A switch can be set to TRUE or FALSE, and the switch-staus can be interrogated programmatically.

General Format:

```
[ {switch-name} [ IS mnemonic-name-1 ]
  {system-name}
[ {ON } STATUS IS switch-status ] ... ] ...
  {OFF}
```

Syntax

1. **mnemonic-name** is a user-defined word.
2. **switch-status** is a user-defined word that represents the ON or OFF status of a program switch.
3. SW0 is an alias of SWITCH-1, SW1 an alias of SWITCH-2, …

General Rules

1. There is a limit of 36 switches that may be set in SPECIAL-NAMES, with ON STATUS and OFF STATUS condition names.
2. [mnemonic-name-n] is used as the target of the SET statement to set the state of a switch.
3. Switch-status is used as the target of the IF statement to test the state of a switch.
4. SW0 to SW15 are supported as aliases of SWITCH-1 to SWITCH-16
5. S01 through S05 are supported.
6. CSP is supported.

Code Sample

```
      *
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  SWITCH-1.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
           SWITCH 1 IS SWITCH-1
             ON STATUS IS SET-CHECK
             OFF STATUS IS SKIP-CHECK.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
           SET SWITCH-1 TO ON.
           IF SET-CHECK
             DISPLAY "SWITCH 1 SET" LINE 10 COL 10
           END-IF.

           DISPLAY "SWITCH-1 FINISHED!" LINE 12 COL 10.
           ACCEPT DUMMY LINE 12 COL 30.
           STOP RUN.
```

ALPHABET

General Format

```
[ {alphabet-entry } ... ]
```

where **alphabet-entry** can have format:

```
ALPHABET alphabet-name IS {NATIVE}
                          {STANDARD-1}
                          {STANDARD-2}
                          {EBCDIC}
```

and **alphabet-entry** may also have format:

```
ALPHABET alphabet-name IS
   {alphabet-literal-1 [ THRU alphabet-literal-2 ] } ...
                     [ {ALSO alphabet-literal-3 } ...]
```

Syntax

1. alphabet-name is a user-defined word.
2. alphabet-literal-n may be either numeric or alphanumeric. When numeric, it is restricted to the range of 1 through 256.

General Rules

1. The Format 1 ALPHABET clause may be used for COLLATING SEQUENCES or character translations.
2. ALPHABET clauses can be used in the following cases:
    a. For COLLATING SEQUENCES
        i. In the SELECT of an INDEXED file for ordering of indexed keys:
            `1. COLLATING SEQUENCE is alphabet-name`
        ii. In a SORT or MERGE statement for ordering of sort/merge keys:
            `1. COLLATING SEQUENCE is alphabet-name`
        iii. In the OBJECT-COMPUTER paragraph for alphanumeric comparisons, and default collating sequence for SORT/MERGE statements:
            `1. PROGRAM COLLATING SEQUENCE is alphabet-name`
    b. For character translations
        i. In the FD of a SEQUENTIAL file:
            `1. CODE-SET is alphabet-name`
3. In an ALPHABET, the first character in the collating sequence is the LOW-VALUES character for that ALPHABET. The last character in the collating sequence is the HIGH-VALUES character for that ALPHABET.
4. The Format 2 ALPHABET clause may be used only for COLLATING SEQUENCES. Characters are listed in the order of their position in the collating sequence.
5. Characters not named are assigned a position greater than the listed characters, in native order.
6. Numeric designations refer to a character's ordinal position in the native character set.

THROUGH/THRU phrase

The THROUGH/THRU phrase designates a set of contiguous characters in the native character set, and assign them successive ascending positions in the alphabet.

ALSO phrase

The ALSO phrase assigns alphabet-literal-1 and alphabet-literal-3 the same position in the collating sequence.

CLASS

CLASS provides a user-defined name with defined parameters that can be used in conditional statements.

General Format

```
[ CLASS class-name IS
       { literal-1 [ {THROUGH} literal-2 ] } ... ] ...
                     {THRU   }
```

Syntax

1. literal-n is a numeric or alphanumeric literal.

General Rules

1. The CLASS clause creates user-defined class-name, and describes the numeric and alphanumeric characters that are included in the CLASS.
2. A data item belongs to the user-defined class-name if it consists entirely of the characters listed in the CLASS clause for class-name.
3. Literal-1 and literal -2 may be numeric, or alphanumeric.
4. When literal-1 and literal-2 are numeric, they represent the ordinal position in the ASCII table of the character that they represent, with the first ordinal position representing null values “00”. As a further example, the number 1 (hex 31, decimal 49) would be represented by the literal 50.
5. When using the THRU phrase, literal-1 and literal-2 must be represented numerically. The THRU phrase causes all of the contiguous characters between the ordinal position in the ASCII table marked by literal-1 and the ordinal position in the ASCII table marked by literal-2 to be included in the CLASS.
6. THRU and THROUGH are synonyms.

Code sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.   class-test.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
           CLASS ALL-VOWELS IS "A", "E", "I", "O", "U", "Y"
           CLASS ALL-DIGITS IS 49 THRU 58.

       DATA DIVISION.

       WORKING-STORAGE SECTION.
       77 TEST-LETTERS  PIC X(6) VALUE "AEIOUY".
       77 TEST-NUMBERS  PIC 9(10) VALUE 1234567890.
       77 WS-DUMMY PIC X.

       PROCEDURE DIVISION.
       0000-MAIN.
           IF TEST-LETTERS IS ALL-VOWELS
                DISPLAY "Test letters are all vowels!" LINE 10 COL 10.
           IF TEST-NUMBERS IS ALL-DIGITS
                DISPLAY "Test numbers are all digits!" LINE 12 COL 10.
           ACCEPT ws-dummy line 12 col 40.
           STOP RUN.
```

CURRENCY SIGN

CURRENCY SIGN designates a 1-character literal to be used in numeric-edited variable declarations containing a currency symbol.

General Format

```
[ CURRENCY SIGN IS currency-literal ]
```

Syntax

1. currency-literal is a one-character alphanumeric literal used to represent a currency.

General Rules

1. In the absence of any CURRENCY SIGN designation, the dollar sign "$" is recognized as the currency symbol.
2. Currency-literal is the currency sign recognized as the currency symbol for purposes of numeric edited variables that contain a currency sign.

CALL-CONVENTION

The CALL-CONVENTION syntax can be used to control which mechanisms the COBOL program should use when calling a subprogram, or which mechanisms it expects to have been used by the program which calls it. Each calling convention you want to use should be declared in the Special-Names paragraph. Here you can assign one of the call convention numbers, defined below, to a user-defined mnemonic name for later use.

General Format

```
[CALL-CONVENTION number IS mnemonic]
```

Syntax

1. number is a numeric literal representing the call convention .
2. mnemonic is a user-defined word used just after the CALL verb to modify the current call convention.

General Rules

The call convention number is a 16-bit number defined as follows:

Bit	Meaning
0-1	Unsupported
2	= 0 - RETURN-CODE is updated on exit

= 1 - RETURN-CODE is not updated on exit

3	= 0 –CALL is resolved dynamically at run time = 1 –CALL is resolved statically at compilation time
4-5	Unsupported
6	Windows Only
	= 0 –CALL use standard C call conventions = 1 –CALL use “STDCALL” WINAPI call conventions (used to call Windows standard API)

Typical values are :

4	Do not modify RETURN-CODE
72	Windows API Call.

CALL-CONVENTION is not supported with ENTRY declarations/

Code sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.      prog2.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
           CALL-CONVENTION 4 IS VOIDCALL.

       ...

       PROCEDURE DIVISION.
       A01.
           MOVE 19 TO RETURN-CODE.
           CALL VOIDCALL "callee" .
      *RETURN-CODE Still equal to 19 what ever callee returns
```

CURSOR

CURSOR returns information about the location of the cursor in an ACCEPT statement, through the variable cursor-pos.

General Format

```
[ CURSOR IS cursor-pos ]
```

Syntax

1. cursor-pos is the name of a numeric data item described in the working-storage section or local-storage section of the program.

General Rules

1. cursor-pos can be declared as a PIC X or PIC 9 field, and can be declared to be either 4 bytes in length, or 6 bytes in length.
2. cursor-pos must be declared in the WORKING-STORAGE SECTION, or LOCAL-STORAGE SECTION of the program.
3. If cursor-pos is 4 bytes in length, bytes 1-2 store the line number of the cursor, and bytes 3-4 store the column number of the cursor.
4. If cursor-pos is 6 bytes in length, bytes 1-3 store the line number of the cursor, and bytes 4-6 store the column number of the cursor.
5. When using SPECIAL NAMES CURSOR and the SCREEN SECTION:
   If the program changes the value of cursor-pos, the next ACCEPT field or screen will move the cursor to the required position.
6. ACCEPT SCREEN attempts to place the the cursor for purposes of input to the value stored in cursor-pos. If the value stored in cursor-pos does not correspond to the location of an input field, the cursor is moved to the best matching field.

CRT STATUS

CRT STATUS returns information about the status of an ACCEPT [Screen] statement when an ON EXCEPTION statement is included in the ACCEPT [Screen] statement.

General Format

```
[ CRT STATUS IS crt-status-var ].
```

Syntax

1. crt-status-var can be described as either an alphanumeric data item described as PIC 9(X) or as a numeric data item described as UNSIGNED-INT, that is declared in the working-storage section.

General Rules

1. The manner in which crt-status-var is updated can depend on the setting of certain environment variables, compiler flags, and compiler configuration flags. See the table below for default behaviors, for associated environment variables:

Condition	Key-pressed	CIT Value of CRT STATUS
Default	[Enter]/[Return]	0
	F1	1001

	F2	1002
	F3	1003
	F4	1004
	F5	1005
	F6	1006
	F7	1007
	F8	1008
	F9	1009
	F10	1010
	F11	1011
	F12	1012
On ACCEPT <Screen> / <Field>	Page Up	2001
where COB_SCREEN_EXCEPTIONS=Y		
	Page Down	2002
	Up Arrow	2003
	Down Arrow	2004
	Print	2006
Where COB_SCREEN_ESC=Y	Esc	2005
Where COB_SCREEN_RAW_KEYS=Y	Home	2007
	End	2008
	Ins	2009
	Del	2010
	Erase EOL	2011

1. Alternatively, it is possible to create your own CRT STATUS Map, using the compiler configuration flag crtstatus-map, as follows:

crtstatus-map: cit-value user-value

2. Note that the manner in which user-value is encoded depends on how the crt-status-var is declared.

When crt-status-var is declared as UNSIGNED-INT, the runtime copies user-value directly into the crt-status-var. For this case, if you wish to remap the first four function keys, from CIT-Values of 1001 through 1004 to single digits 1 through 4, you would add the entries:

crtstatus-map: 1001 1

crtstatus-map: 1002 2

crtstatus-map: 1003 3

crtstatus-map: 1004 4

Note that the use of hex notation (x00 through x127) to describe user-value is supported. For this case, if you wish to remap the first four function keys, from CIT-Values of 1001 through 1004 to single digits 1 through 4, you would add the entries:

crtstatus-map: 1001 x31

crtstatus-map: 1002 x32

crtstatus-map: 1003 x33

crtstatus-map: 1004 x34

Hex notations permit the user to use values that fall outside of the normal alphanumeric range. More than one character can be represented using hex notation. For example, if you wish to map the cit-value of 1004 to the user-value of 1234 using hex notation, you would add the entry:

crtstatus-map 1004 x31323334

In the example above, note that in the description of theuser-value, after the “x”, the values “31”, “32”, “33”, “34” are concatenated to represent the string “1234”. When using hex notations, it is recommended that the receiving field be described as PIC X(4).

If no crtstatus-map is defined , CRT STATUS values are converted to PIC 9(4) and copied into the crt-status-var.

screenio.cpy

The file screenio.cpy is installed in the %COBOLITDIR%\copy folder in Windows and the $COBOLITDIR/share/cobol-it/copy directory for Linux/Unix.. screenio.cpy contains a number of level-78 constant declarations that are useful in screen I-O handling, including constant declarations for the default CRT-STATUS values that are returned by function keys, and on certain screen errors, in addition to screen color values.

The contents of screenio.cpy:

```
*> Colors as defined by the Cobol standard
 78  COB-COLOR-BLACK     VALUE 0.
 78  COB-COLOR-BLUE      VALUE 1.
 78  COB-COLOR-GREEN     VALUE 2.
 78  COB-COLOR-CYAN      VALUE 3.
 78  COB-COLOR-RED       VALUE 4.
 78  COB-COLOR-MAGENTA   VALUE 5.
 78  COB-COLOR-YELLOW    VALUE 6.
 78  COB-COLOR-WHITE     VALUE 7.
*>
*> Values that may be returned in CRT STATUS (or COB-CRT-STATUS)
*> Normal return - Value 0000
 78  COB-SCR-OK          VALUE 0.
*>  Function keys - Values 1xxx
 78  COB-SCR-F1          VALUE 1001.
 78  COB-SCR-F2          VALUE 1002.
 78  COB-SCR-F3          VALUE 1003.
 78  COB-SCR-F4          VALUE 1004.
 78  COB-SCR-F5          VALUE 1005.
 78  COB-SCR-F6          VALUE 1006.
 78  COB-SCR-F7          VALUE 1007.
 78  COB-SCR-F8          VALUE 1008.
 78  COB-SCR-F9          VALUE 1009.
 78  COB-SCR-F10         VALUE 1010.
 78  COB-SCR-F11         VALUE 1011.
 78  COB-SCR-F12         VALUE 1012.
 78  COB-SCR-F13         VALUE 1013.
 78  COB-SCR-F14         VALUE 1014.
 78  COB-SCR-F15         VALUE 1015.
 78  COB-SCR-F16         VALUE 1016.
 78  COB-SCR-F17         VALUE 1017.
 78  COB-SCR-F18         VALUE 1018.
 78  COB-SCR-F19         VALUE 1019.
 78  COB-SCR-F20         VALUE 1020.
 78  COB-SCR-F21         VALUE 1021.
 78  COB-SCR-F22         VALUE 1022.
 78  COB-SCR-F23         VALUE 1023.
 78  COB-SCR-F24         VALUE 1024.
 78  COB-SCR-F25         VALUE 1025.
 78  COB-SCR-F26         VALUE 1026.
 78  COB-SCR-F27         VALUE 1027.
 78  COB-SCR-F28         VALUE 1028.
 78  COB-SCR-F29         VALUE 1029.
 78  COB-SCR-F30         VALUE 1030.
```

```
 78  COB-SCR-F31          VALUE 1031.
 78  COB-SCR-F32          VALUE 1032.
 78  COB-SCR-F33          VALUE 1033.
 78  COB-SCR-F34          VALUE 1034.
 78  COB-SCR-F35          VALUE 1035.
 78  COB-SCR-F36          VALUE 1036.
 78  COB-SCR-F37          VALUE 1037.
 78  COB-SCR-F38          VALUE 1038.
 78  COB-SCR-F39          VALUE 1039.
 78  COB-SCR-F40          VALUE 1040.
 78  COB-SCR-F41          VALUE 1041.
 78  COB-SCR-F42          VALUE 1042.
 78  COB-SCR-F43          VALUE 1043.
 78  COB-SCR-F44          VALUE 1044.
 78  COB-SCR-F45          VALUE 1045.
 78  COB-SCR-F46          VALUE 1046.
 78  COB-SCR-F47          VALUE 1047.
 78  COB-SCR-F48          VALUE 1048.
 78  COB-SCR-F49          VALUE 1049.
 78  COB-SCR-F50          VALUE 1050.
 78  COB-SCR-F51          VALUE 1051.
 78  COB-SCR-F52          VALUE 1052.
 78  COB-SCR-F53          VALUE 1053.
 78  COB-SCR-F54          VALUE 1054.
 78  COB-SCR-F55          VALUE 1055.
 78  COB-SCR-F56          VALUE 1056.
 78  COB-SCR-F57          VALUE 1057.
 78  COB-SCR-F58          VALUE 1058.
 78  COB-SCR-F59          VALUE 1059.
 78  COB-SCR-F60          VALUE 1060.
 78  COB-SCR-F61          VALUE 1061.
 78  COB-SCR-F62          VALUE 1062.
 78  COB-SCR-F63          VALUE 1063.
 78  COB-SCR-F64          VALUE 1064.
*>  Exception keys - Values 2xxx
 78  COB-SCR-PAGE_UP      VALUE 2001.
 78  COB-SCR-PAGE_DOWN    VALUE 2002.
 78  COB-SCR-KEY-UP       VALUE 2003.
 78  COB-SCR-KEY-DOWN     VALUE 2004.
 78  COB-SCR-ESC          VALUE 2005.
 78  COB-SCR-PRINT        VALUE 2006.
 78  COB-SCR-HOME-KEY     VALUE 2007.
 78  COB-SCR-END-KEY      VALUE 2008.
 78  COB-SCR-INS-KEY      VALUE 2009.
 78  COB-SCR-DEL-KEY      VALUE 2010.
 78  COB-SCR-FWD-TAB      VALUE 9.
 78  COB-SCR-BWD-TAB      VALUE 2012.
 78  COB-SCR-KEY-LEFT     VALUE 2013.
 78  COB-SCR-KEY-RIGHT    VALUE 2014.
*>  Input validation - Values 8xxx
 78  COB-SCR-NO-FIELD     VALUE 8000.
```

```
*>  Other errors - Values 9xxx
 78  COB-SCR-FATAL       VALUE 9000.
 78  COB-SCR-TIMEOUT     VALUE 9001.
```

NUMERIC SIGN

The NUMERIC SIGN clause is used to define a default sign storage convention, when the sign storage for a data item is not specified.

General Format

```
[NUMERIC [SIGN [IS] [ (LEADING }   [ SEPARATE ]
                      {TRAILING}
```

General Rules

1. The default sign storage may be set to LEADING, TRAILING, LEADING SEPARATE, or TRAILING SEPARATE. For details on the internal data storage, see 4.8.15 Sign Clause.

DECIMAL-POINT

DECIMAL-POINT converts decimal-points associated with numeric edited data items to commas, and converts commas associated with numeric edited data items to decimal-points.

General Format

```
[ DECIMAL-POINT IS COMMA ]
```

General Rules

1. Numeric edited data items convert commas to decimal points, and decimal points to commas.

3.1.4 REPOSITORY.

The REPOSITORY paragraph lists the intrinsic-function-names that may be used in the program without specifying the word FUNCTION.

General Format

```
REPOSITORY.
    [ FUNCTION { { intrinsic-function-name } ... } INTRINSIC ].
               { ALL                             }
```

Syntax

1. intrinsic-function-name is the name of an intrinsic function that may be used without specifying the word FUNCTION. For a full list of intrinsic function names, see Appendix 6.4.

General Rules

1. FUNCTION ALL indicates that all intrinsic functions may be used without specifying the word FUNCTION.
2. The compiler flag –ffunctions-all has the same effect as the FUNCTION ALL clause, allowing all intrinsic functions to be used without specifying the word FUNCTION.
3. In the absence of a designation in the REPOSITORY paragraph, or the use of the –ffunctions-all compiler flag, the FUNCTION keyword is required to use an intrinsic function. If it is not used, an error, such as : **Error: 'SQRT' undefined** is returned by the compiler.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.   repos-test.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       REPOSITORY.
           FUNCTION ALL INTRINSIC.
       DATA DIVISION.

       WORKING-STORAGE SECTION.
       77 WS-DUMMY PIC X.
       77 sqrt-fld PIC 99.
       PROCEDURE DIVISION.
       MAIN.
           MOVE SQRT(225) TO sqrt-fld.
           DISPLAY "Square Root of 225 is: " line 10 col 10.
           DISPLAY sqrt-fld line 10 col 33.
           ACCEPT ws-dummy line 10 col 40.

           STOP RUN.
```

3.2 INPUT-OUTPUT SECTION.

The INPUT-OUTPUT Section describes the Input-Output environment of the program. When compiling with a relaxed syntax check, the header for this section is optional.

General Format

```
[ [ INPUT-OUTPUT SECTION. ]
  [ FILE-CONTROL. ] { file-control-entry } ...
  [ I-O-CONTROL. [ i-o-control-entry ] ] ] ]
```

General Rules

1. The General Rules for the Statements and Clauses of the Environment Division entries are described below.

3.2.1 FILE-CONTROL Paragraph.

The FILE-CONTROL paragraph contains information about the files in the program.

General Format

```
[ FILE-CONTROL. ] { file-control-sequence } ...

    SELECT [OPTIONAL} filename
    select_clause_sequence '.'
```

Syntax

1. The **file-control-sequence** is a description of the attributes of an Indexed, Sequential, Relative, or Sort file.
2. The **select_clause_sequence** is a sequence of clauses, as described below, which provide the information needed to create, and manage the named file.
3. Not all clauses are valid in all file formats. Below are the valid usages of clauses in Indexed, Sequential, Relative, and Sort files respectively:

General Rules

1. The General Rules for each of the different file formats is described below.

INDEXED FILE FORMAT.

General Format

```
SELECT [OPTIONAL] file-name
      ASSIGN TO [ DYNAMIC ] [ { DISK    } ] [ assignment-name ]
                [ EXTERNAL]

    [ ORGANIZATION IS ] INDEXED

    [ ACCESS MODE IS { SEQUENTIAL } ]
                     { RANDOM     }
                     { DYNAMIC    }
```

```
    [ RECORD KEY IS record-key ]
    [ ALTERNATE RECORD KEY IS alt-rec-key [WITH [NO] DUPLICATES] ]

    [ LOCK MODE IS {MANUAL    WITH LOCK ON [MULTIPLE] {RECORD }} ]
                                                      {RECORDS}
                   {AUTOMATIC WITH LOCK ON [MULTIPLE] {RECORD }}
                                                      {RECORDS}
                   {EXCLUSIVE                                  }
                   [ WITH ROLLBACK ]

    [ SHARING [WITH] { ALL OTHER } ].
                     { NO OTHER  }
                     { READ ONLY }

    [ RESERVE {number} [AREA ]]
              {NO    } [AREAS]
    [ COLLATING SEQUENCE IS alphabet-name ]

    [ FILE STATUS IS file-status-var1 ] .
```

SEQUENTIAL FILE FORMAT.

General Format

```
SELECT [OPTIONAL} filename

     ASSIGN TO [ DYNAMIC ] [ { DISK    } ] [ assignment-name ]
               [ EXTERNAL]   { PRINTER }

     [ [ ORGANIZATION IS ] {[BINARY] SEQUENTIAL} ]
                           [RECORD]
                           [LINE ]

     [ ACCESS MODE IS SEQUENTIAL ]

     [ LOCK MODE IS { MANUAL    }  ]
                    { AUTOMATIC }
                    { EXCLUSIVE }

     [ SHARING [WITH] { ALL OTHER } ].
                      { NO OTHER  }
                      { READ ONLY }

     [ RESERVE {number} [AREA ]]
               {NO    } [AREAS]

     [ RECORD DELIMITER IS STANDARD-1 ]

     [ FILE STATUS IS file-status-var1 ]

     [ PADDING CHARACTER IS padding-char ].
```

RELATIVE FILE FORMAT.

General Format

```
SELECT [OPTIONAL} filename
      ASSIGN TO [ DYNAMIC ] [ { DISK    } ] [ assignment-name ]
                [ EXTERNAL]

    [ ORGANIZATION IS ] RELATIVE

    [ ACCESS MODE IS { SEQUENTIAL } ]
                     { RANDOM     }
                     { DYNAMIC    }

    [ RELATIVE KEY IS relative-key ]

    [ LOCK MODE IS {MANUAL    WITH LOCK ON [MULTIPLE] {RECORD }} ]
                                                      {RECORDS}
                  {AUTOMATIC WITH LOCK ON [MULTIPLE] {RECORD }}
                                                     {RECORDS}
                  {EXCLUSIVE                                 }

    [ SHARING [WITH] { ALL OTHER } ].
                     { NO OTHER  }
                     { READ ONLY }

    [ RESERVE {number} [AREA ]]
              {NO    } [AREAS]

    [ FILE STATUS IS file-status-var1 ] .
```

SORT FILE FORMAT.

General Format

```
 SELECT file-name
      ASSIGN TO [ DYNAMIC ] [ { DISK    } ] [ assignment-name ]
                [ EXTERNAL]

     [ FILE STATUS IS file-status-var1 ] .
```

File Control Clauses

ASSIGN Clause

The ASSIGN clause describes where the data will be stored.

General Format

```
     ASSIGN TO [ DYNAMIC ] [ { DISK    } ] [ assignment-name ]
               [ EXTERNAL]   { PRINTER }
```

Syntax

1. assignment-name is either a non-numeric literal or an alphanumeric data item in the working-storage section.
2. File-name and assignment-name may be the same name.

ASSIGN Clause General Rules -

1. The DYNAMIC phrase indicates that assignment-name is a data item. assignment-name does not need to be initialized to a value when the run unit is initiated, but it must have a value when the OPEN statement is executed.
2. The EXTERNAL phrase indicates that assignment-name is a COBOL word. COBOL-IT will look for a file of the exact name, or an environment variable that matches assignment-name to resolve the filename.
3. The DISK and PRINTER phrases can only be used with files described with sequential files, as determined by the ORGANIZATION clause.
4. The PRINTER phrase causes the file to be considered a Print file. Print files can only be OPENed OUTPUT, and EXTEND. Records that written to a Print files strip trailing spaces. See “Using data files” in the Getting Started with COBOL-IT document for more information on how COBOL-IT resolves filenames included in the ASSIGN clause.
5. Unix command pipes are not supported by COBOL-IT. Notations such as "| cmd" or "< cmd " in a file name are not supported. If this functionality is required, you should use the use CALL "SYSTEM", write output to a temporary file, and then address the temporary file in your ASSIGN Clause.

LOCK MODE Clause

General Format

```
[LOCK MODE IS     {MANUAL    WITH LOCK ON [MULTIPLE] {RECORD }} ]
                                                    {RECORDS}

                  {AUTOMATIC WITH LOCK ON [MULTIPLE] {RECORD }}
                                                    {RECORDS}

                  {EXCLUSIVE                                  }

                  [ WITH ROLLBACK ]
```

LOCK MODE Clause General Rules

1. The file system must support COMMIT and ROLLBACK, otherwise the WITH ROLLBACK clause, COMMIT statement and ROLLBACK statement have no effect.
2. When the WITH ROLLBACK phrase is used in the LOCK MODE clause, a file may use the COMMIT and ROLLBACK statements.

3. WITH ROLLBACK implies LOCK MODE IS AUTOMATIC, as any of the I-O statements WRITE, REWRITE, DELETE automatically place a lock on the record that is the target of the I-O statement. These locks are released when either the COMMIT or ROLLBACK statement is executed.
4. The effect of the ROLLBACK statement is that all record updates, through WRITE, REWRITE, DELETE statements since the last COMMIT or ROLLBACK statement are nullified, or "rolled back".
5. The effect of the COMMIT statement is that all record updates, through WRITE, REWRITE, DELETE statements are flushed to from buffers to the data source, and locks released.

ACCESS Clause

The ACCESS clause describes whether the data will be accessible sequentially, or through keyed READs, or both.

General Format

```
[ACCESS [MODE] IS { SEQUENTIAL } ]
                  { RANDOM     }
                  { DYNAMIC    }
```

ACCESS Clause General Rules

1. ACCESS MODE IS SEQUENTIAL indicates that records are only accessible sequentially, through READ NEXT statements, which retrieve the next record, as defined by the organization of the file.
2. ACCESS MODE IS SEQUENTIAL can be applied to files described with ORGANIZATION that is SEQUENTIAL, RELATIVE, or INDEXED.
3. ACCESS MODE IS RANDOM indicates that records are only accessible through READs on a KEY.
4. ACCESS MODE IS RANDOM can be applied to files described with ORGANIZATION that is RELATIVE or INDEXED. For relative files, the RELATIVE KEY is the key of reference. For indexed files, the RECORD KEY is the key of reference.
5. ACCESS MODE IS DYNAMIC indicates that records are accessible either through READ NEXT statements, or READs on a KEY.
6. ACCESS MODE IS DYNAMIC can be applied to files described with ORGANIZATION that is RELATIVE or INDEXED.
7. If no ACCESS MODE is described, then ACCESS MODE IS SEQUENTIAL is assumed.

ALTERNATE RECORD KEY Clause

The ALTERNATE RECORD KEY clause describes a non-primary RECORD KEY which may or may not accept DUPLICATES.

General Format

```
[ ALTERNATE RECORD KEY IS alt-rec-key [WITH [NO] DUPLICATES] ]

[ ALTERNATE RECORD KEY IS split-key-name = item1, item2 …
                                        [WITH [NO] DUPLICATES] ]
```

Syntax

1. alt-rec-key is a data item in the file's FD record description.
2. split-key-name must be unique in the module.

ALTERNATE RECORD KEY Clause General Rules

1. The number of alternate record keys in a file is set when the file is created.

alt-rec-key names a data element within the file description (FD) as an alternate key of reference for the file.

2. split-key-name is a user-defined name that is associated with a series of non-contiguous data elements within the file description as a key of reference. Split-key-name is needed when using split keys, and referencing the non-contiguous data elements in a READ KEY IS [record-key] or START KEY IS [record-key] statement.
3. The WITH DUPLICATES phrase indicates that the data file may contain more than one record matching the given alternate record key of reference.
4. The WITH NO DUPLICATES phrase indicates that the alternate record key of reference is a unique identifier of a record within the file.

Collating Sequence Clause

The COLLATING SEQUENCE clause names the alphabet used for purposes of sorting a key in an indexed file. The alphabet must be declared in SPECIAL-NAMES.

```
[ COLLATING SEQUENCE IS alphabet-name ]
```

COLLATING SEQUENCE Clause General Rules

1. The alphabet-name used in the COLLATING SEQUENCE clause must be either an alphabet-name that is declared in SPECIAL-NAMES with an ALPHABET clause, or EBCDIC.

FILE STATUS Clause

The FILE STATUS clause returns the status of the most recent FILE I/O operation.

General Format

```
[ FILE STATUS IS file-status-var ]
```

Syntax

1. file-status-var may be declared as PIC 99 or PIC XX.

FILE STATUS Clause General Rules

1. file-status-var is updated following every I-O operation for the associated file.
2. A status of "00" indicates a successful I-O operation.
3. For information on how unsuccessful I-O operations update file-status-var, see 5.6 File Status Codes.

LOCK MODE Clause

The LOCK MODE clause describes the means by which LOCKing of records will be applied within the File.

General Format

```
[LOCK MODE IS {MANUAL    WITH LOCK ON [MULTIPLE] {RECORD }} ]
                                                 {RECORDS}
              {AUTOMATIC WITH LOCK ON [MULTIPLE] {RECORD }}
                                                 {RECORDS}
              {EXCLUSIVE                                  }

              {WITH ROLLBACK }
```

LOCK MODE Clause General Rules

1. If there is no LOCK MODE clause, then LOCK MODE is AUTOMATIC is assumed unless a SHARING clause in the SELECT statement or in the OPEN statement indicates otherwise. Note that some file systems and some operating systems do not support record locking. In these cases, all references to LOCK MODE are treated as commentary.
2. The LOCK MODE IS MANUAL phrase applies when there is a READ on a record in a file that is OPENed in the I-O mode. When LOCK MODE IS MANUAL, the READ statement must include a WITH LOCK clause in order for a lock to be applied.
3. The LOCK MODE IS AUTOMATIC phrase applies when there is a READ on a record in a file that is OPENed in the I-O mode. When LOCK MODE IS AUTOMATIC, the record retrieved by the READ is locked automatically, unless the READ statement includes a WITH NO LOCK clause.
4. THE LOCK MODE IS EXCLUSIVE phrase locks the entire file for exclusive use.
5. The WITH LOCK ON MULTIPLE RECORDS phrase allows the file to hold multiple record locks within the same file.
6. The WITH ROLLBACK phrase is recognized, and syntax is validated. However, the WITH ROLLBACK clause is otherwise treated as commentary.

LINE ADVANCING Clause

1. The LINE ADVANCING clause is recognized, and syntax is validated. However, the LINE ADVANCING clause is otherwise treated as commentary.

LINE ADVANCING Clause General Rules

There are no General Rules.

OPTIONAL Clause

The OPTIONAL clause affects the behaviour of the OPEN statement when the file being OPEN'ed is not present.

OPTIONAL Clause General Rules

1. When the OPTIONAL clause is included in the SELECT phrase, the OPEN statement does not require that the named file be available in order for the OPEN to be successful.
2. The effect of the OPTIONAL clause varies depending on the OPEN mode.
   a. OPEN INPUT- The OPEN succeeds, but the file is not created. The first attempt to READ data in the file fails.
   b. OPEN I-O- The OPEN succeeds and the file is created. WRITEs to the file are successful.
   c. OPEN Extend- The OPEN succeeds and the file is created. WRITEs to the file are successful.

ORGANIZATION Clause

The ORGANIZATION clause describes the logical structure of the file.

General Format

```
[ [ ORGANIZATION IS ] {INDEXED            }  ]
                      {[BINARY] SEQUENTIAL}
                       [RECORD]
                       [LINE ]
                      {RELATIVE           }
```

ORGANIZATION Clause General Rules

1. ORGANIZATION IS INDEXED phrase indicates that the file is an indexed file. Indexed files can be accessed sequentially, or through a RECORD KEY, which identifies a record, and which establishes the order in which the records are ordered in the file.
2. The ORGANIZATION IS RELATIVE phrase indicates that the file is a relative file. Relative files can be accessed sequentially, or through a RELATIVE KEY, which represents a record's ordinal position in the file.
3. ORGANIZATION IS SEQUENTIAL indicates that the file is a sequential file. Sequential files can only be accessed sequentially. Records are stored in a sequential file in the order in which they were added to the file.
4. ORGANIZATION IS BINARY/RECORD SEQUENTIAL indicates that the file is a

sequential file, and does not contain record delimiters.

5. BINARY and RECORD are synonyms.
6. ORGANIZATION IS LINE SEQUENTIAL indicates that the file is a sequential file, and contains record delimiters.

PADDING CHARACTER Clause

The PADDING CHARACTER designates a single character literal to pad the last block of the file.

General Format

```
[ PADDING CHARACTER IS padding-char ]
```

Syntax

1. padding-char is a single character literal or alphanumeric data item.

PADDING CHARACTER Clause General Rules

1. If padding-char is not specified, the SPACE is used as padding-char.
2. When a PADDING CHARACTER is defined, that character is used to pad the last block of the file.

RECORD DELIMITER Clause

The RECORD DELIMITER clause is recognized, and syntax is validated. However, the RECORD DELIMITER clause is otherwise treated as commentary.

General Format

```
[ RECORD DELIMITER IS STANDARD-1 ]
```

RECORD DELIMITER Clause General Rules

There are no General Rules.

RECORD KEY Clause

The RECORD KEY clause names the field in the file definition of an indexed file that is to be considered the primary key, for purposes of RANDOMly accessing records.

General Format

```
[ RECORD KEY IS record-key ]

[ RECORD KEY IS split-key-name = item1 item2 … ]
```

Syntax

1. record-key is a data item in the file's FD record description.
2. split-key-name must be unique in the module

RECORD KEY Clause General Rules

1. RECORD KEY is defined when the file is created.
2. Record-key names a data element within the file description (FD) as a primary key of reference for the file, determining the order in which records are retrieved when READ using RECORD KEY.
3. RECORD KEY is a unique identifier of a record within the file.
4. split-key-name is a user-defined name that is associated with a series of non-contiguous data elements within the file description as a key of reference. Split-key-name is needed when using split keys, and referencing the non-contiguous data elements in a READ KEY IS [record-key] or START KEY IS [record-key] statement.

RELATIVE KEY Clause

The RELATIVE KEY clause names the field in the working-storage section that is to be considered the relative key, for purposes of RANDOMly accessing records in a relative file.

General Format

```
[ RELATIVE KEY IS rel-key ]
```

Syntax

1. rel-key is an unsigned integer data item declared in the working-storage section.

RELATIVE KEY Clause General Rules

1. Relative files can be accessed through a RELATIVE KEY, which is stored in an unsigned data item declared in the working-storage section, and which represents a record's ordinal position in the file.
2. rel-key must be set to a positive integer value before performing I-O operations.

RESERVE Clause

The RESERVE clause is recognized, and syntax is validated. However, the RESERVE clause is otherwise treated as commentary.

General Format

```
[ RESERVE {number} [AREA ]]
          {NO }    [AREAS]
```

RESERVE Clause General Rules

There are no General Rules.

SHARING Clause

The SHARING clause describes the situations in which concurrent access to a file is allowed from multiple processes.

General Format

```
[ SHARING [WITH] { ALL OTHER } ].
                 { NO OTHER  }
                 { READ ONLY }
```

SHARING Clause General Rules

1. The SHARING clause sets a level of restrictiveness on whether a file that has been OPENed can be OPENed by another file connector. Conditions causing a failed OPEN are marked with the word "Fails" , and conditions leading to a successful OPEN are marked with the word "Succeeds" in the table below:

	Sharing WITH NO OTHER	Sharing WITH READ ONLY	Sharing WITH ALL OTHER
OPEN	EXTEND, I-O,	EXTEND, INPUT I-O,	EXTEND, INPUT I-O,

Sharing /	MODE	INPUT, OUTPUT	OUTPUT		OUTPUT	
	EXTEND, I-O,					
WITH NO OTHER	INPUT , OUTPUT	Fails	Fails	Fails	Fails	Fails
WITH READ ONLY	EXTEND, I-O	Fails	Fails	Fails	Fails	Succeeds
	INPUT	Fails	Fails	Succeeds	Fails	Succeeds
	OUTPUT	Fails	Fails	Fails	Fails	Fails
WITH ALL OTHER	EXTEND, I-O	Fails	Fails	Fails	Succeeds	Succeeds
	INPUT	Fails	Succeeds	Succeeds	Succeeds	Succeeds
	OUTPUT	Fails	Fails	Fails	Fails	Fails

2. A SHARING phrase on an OPEN statement takes precedence over a SHARING phrase in a SELECT statement.

3.2.2 I-O-CONTROL.

The I-O-CONTROL paragraph describes input-output rules that are applicable for specified files.

General Format

```
I-O-CONTROL.
     [ APPLY WRITE-ONLY ON {file-name} ... ] ...
     [ SAME [RECORD ] AREA FOR {file-name} ... ] ...
           [SORT ]
           [SORT-MERGE]
     [ MULTIPLE FILE TAPE CONTAINS
     { file-name [ POSITION position-integer ] } ... ] ... .
```

Syntax

1. file-name is a file described by a SELECT clause in the FILE-CONTROL paragraph.
2. position-integer is an integer literal.

General Rules

1. The APPLY clause is recognized, and syntax is validated. However, the APPLY clause is otherwise treated as commentary.
2. The SAME RECORD AREA clause causes the listed files to share the same memory. Care should be taken to ensure that of the listed files, only one is OPEN at a time.
3. The SAME SORT AREA clause is recognized, and syntax is validated. However, the

SAME SORT AREA clause is otherwise treated as commentary.

4. The SAME SORT-MERGE AREA clause is recognized, and syntax is validated. However, the SAME SORT-MERGE AREA clause is otherwise treated as commentary.
5. The MULTIPLE FILE TAPE clause is recognized, and syntax is validated. However, the MULTIPLE FILE TAPE clause is otherwise treated as commentary.

4 DATA DIVISION.

The DATA DIVISION contains the descriptions of all of the data items that are used in the program. The separate SECTIONs of the Data Division separate the data descriptions, according to functionality:

General Format

```
DATA DIVISION.
[ FILE SECTION. ]
  [ file-descripiton  ]…
  [ fd-data-level fd-data-description ] …
  [ sort-description  ]…
  [ sd-data-level sd-data-description ] …
[ WORKING-STORAGE SECTION. ]
  [ ws-data-level ws-data-description ] …
[ LOCAL-STORAGE SECTION. ]
  [ ls-data-level ls-data-description ] …
[ LINKAGE SECTION. ]
  [ lk-data-level lk-data-description ] …
[ SCREEN SECTION. ]
  [ ss-data-level screen-item-description]…
```

Syntax

1. file-description is descriptions of data items used in sequential, relative, and indexed files.
2. fd-data-level is a number between 01 and 49 (inclusive), 66, 78, or 88.
3. fd-data-description is descriptions of data items used in sequential, relative, and indexed, files.
4. sort-description is descriptions of data items used in sort files.
5. sd-data-level is a number between 01 and 49 (inclusive), 66, 78, or 88.
6. sd-data-description is descriptions of data items used in sort files.
7. ws-data-level is a number between 01 and 49 (inclusive), 66, 77, 78, or 88.
8. ws-data-description is descriptions of data items used by the program internally.
9. ls-data-level is a number between 01 and 49 (inclusive), 66, 77, 78, or 88.
10. ls-data-description is descriptions of data items used by nested programs
11. lk-data-level is a number between 01 and 49 (inclusive), 66, 77, 78, or 88.
12. lk-data-description is is descriptions of data items that have been passed to a called subprogram
13. ss-data-level is a number between 01 and 49 (inclusive).
14. ss-data-description is descriptions of Screen Section console screen items.

General Rules

The General Rules for the Data Division Clauses are documented below.

4.1 Data Level Numbers

General Rules

Data level numbers are numbers between 01 and 49 (inclusive), 66, 77, 78, or 88.

Level 01 thru 49

1. The 01-level is the top level of a data description.
2. Data items with data levels between 01 and 49 (inclusive) may be divided into data elements containing data level numbers greater than the data level of the parent data item, and less than or equal to 49. Note that level 66, 78, and 88 data levels are permitted in a group data item, but they are each special cases, as described below.
3. Level-77 data items may not be used in a group data item.

Example:

```
01 customer-address.
  05 customer-street-number     pic 9(7).
  05 customer-street-name       pic x(30).
  05 customer-apt-num           pic 9(3).
```

1. The top data level of a record description in an FD must be an 01-level.
2. It is possible for a file to have multiple record descriptions. This is indicated with multiple 01-level record descriptions.

Level 66

1. Level 66 is used with the RENAMES clause, and permits the programmer to rename a data element, or series of contiguous data elements immediately preceding the declaration of the level 66 data item.

Example:

```
01 group-item.
  05 customer-name    pic x(30) .
  05 customer-address pic x(30).
```

```
66 customer-info renames customer-name thru customer address.
```

Level 77

1. Level 77 data items may not be used in record descriptions in the File Section.
2. Level 77 data items may not be divided into data elements, nor may level 77 data items be used in a group item.
3. Level 77 data items may have level 88 conditions associated with them.

Example:

```
77 file-status    pic xx.
```

Level 78

1. Level 78 data items equate a data name with a numeric or alphanumeric constant. This can add readability to source code.

Example:

```
78 one-thousand      value 1000.
78 largest-planet    value "Jupiter".
```

This permits code such as :

ADD one-thousand TO year-end-bonus.

DISPLAY largest-planet LINE 10 COL 10.

Level 88

1. Level 88 data items describe condition names. This can add readability to source code.

Example:

General Format

```
77 file-status pic xx.
  88 end-of-file  value "10".
```

This allows for the replacement of code such as :

If file-status = “10”

with

If end-of-file

4.2 Reference Modification

Reference Modification provides syntax for referencing a portion of the memory allocated by a data item.

General Format

[data-name] (starting-offset : [number-of-bytes])

Syntax

1. Data-name is a data-item declared in the Data Division.
2. Starting-offset is a numeric literal, or data-item.

General Rules

1. Starting-offset corresponds to the location in data-name where bytes start to be referenced. A value of 1 corresponds to the first byte.
2. Number-of-bytes is a numeric literal, or data item. Number-of-bytes corresponds to the number of bytes that are to be referenced beginning at starting-offset.
3. Number-of-bytes must be less than [length-of-data-name] – [starting-offset] + 1.
4. Number-of-bytes is optional. If number-of-bytes is not included in the reference modification phrase, then it is implied that the data referenced begins at starting-offset, and continues to the last byte in data-name.
5. The substring identified by reference modification is treated as an alphanumeric data item, and a reference modification can be used anywhere that an alphanumeric literal or data item may be used.

Examples:
For the string:
77 white-house-address PIC X(21) VALUE “1600 Pennsylvania Ave”.

White-house-address(1:4) returns “1600”.
White-house-address(19:) returns “Ave”.

Reference modification can be applied to the output of an intrinsic function:

FUNCTION CURRENT-DATE (5:2) returns the current month.

4.3 FILE SECTION.

The File Section contains the descriptions of data items used in sequential, relative, indexed, and sort files.

General Format

```
[ FILE SECTION. ]
 [ file-descripiton  ]…
 [ fd-data-level fd-data-description ] …
 [ sort-description  ]…
 [ sd-data-level sd-data-description ] …
```

The General Formats for the File Description and Sort Description are below:

General Rules

1. The File Section contains the File Descriptions and Sort Descriptions.
2. Every File Description and Sort Description must have a corresponding SELECT statement in the Environment Division.

4.3.1 FILE DESCRIPTION.

The File Descripiton describes the characteristics of a sequential, relative, or indexed file.

General Format

```
FD  file-name [ IS { EXTERNAL } ]
                   { GLOBAL   }

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

     [ CODE-SET IS alphabet-name ]

     [RECORD { CONTAINS int-1 [TO int-2 ] CHARACTERS } ]
     [ IS VARYING IN SIZE [ FROM int-3 ] [ TO int-4 ] CHARACTERS ]
     [ DEPENDING ON record-depending ]
     [ MODE IS VARIABLE [RECORD VARYING FROM 1]]



     [ LABEL RECORDS [ARE] {STANDARD} ]
                           {OMITTED }
```

```
      [ VALUE OF { LABEL   } IS valueof-name ]
                 { FILE-ID }

      [ { DATA RECORD IS } {record-name} ... ]
        { DATA RECORDS ARE }

      [ LINAGE IS page-size LINES
      [ WITH FOOTING AT footing-line ]
      [ LINES AT TOP top-lines ]
      [ LINES AT BOTTOM bottom-lines ] ] .

       [ RECORDING MODE IS { F }                          ]
                           { V }
                           { U }
                           { S }
        [ { REPORT IS   } report-name ]
          { REPORTS ARE }
```

General Rules

1. file- name described in a File Description (FD), may be referenced by OPEN, CLOSE, START, READ, and UNLOCK statements.
2. The General Rules for the File Description clauses are described below.

4.3.2 SORT DESCRIPTION.

The SORT Description describes the characteristics of a SORT file.

General Format

```
SD  file-name
     [RECORD { CONTAINS int-1 [TO int-2 ] CHARACTERS } ]
     [ IS VARYING IN SIZE [ FROM int-3 ] [ TO int-4 ] CHARACTERS ]
     [ DEPENDING ON record-depending ]

     [ { DATA RECORD IS } {record-name} ... ]
       { DATA RECORDS ARE }

     [ VALUE OF FILE-ID IS valueof-name ]
```

General Rules

1. file- name described in a Sort Description (SD), may only be referenced by SORT and MERGE statements.

The General Rules for the Sort Description clauses are described below.

4.3.3 FILE SECTION Clauses

BLOCK CONTAINS Clause

The BLOCK CONTAINS clause is recognized, and syntax is validated. However, the BLOCK CONTAINS clause is otherwise treated as commentary.

General Format

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

Syntax

1. integer-1 is an integer
2. integer-2 is an integer that is greater than integer-1

General Rules

There are no General Rules.

CODE-SET Clause

The CODE-SET clause associates an *ALPHABET* with a sequential file.

General Format

```
[ CODE-SET IS alphabet-name ]
```

Syntax

1. The CODE-SET clause may only be applied to SEQUENTIAL files.
2. *Alphabet-name* is the name of an alphabet described in SPECIAL-NAMES.
3. The CODE-SET clause requires that all data items in the record description be USAGE DISPLAY.

Code Set Clause General Rules

1. The CODE-SET clause uses *alphabet-name* to represent the character set in which the file data is stored.
2. Usage of the CODE-SET clause causes data conversion between the native character set and the character set described by *alphabet-name* to occur on input and output File I/O operations.

DATA RECORDS Clause

The DATA RECORDS clause names the 01-level record names in a file.

The DATA RECORDS clause is is recognized, and syntax is validated. However, the DATA RECORDS clause is otherwise treated as commentary.

General Format

```
[ { DATA RECORD IS   } {record-name} ... ]
  { DATA RECORDS ARE }
```

Syntax

1. record-name(s) are the 01-level record names of the file.

General Rules

There are no General Rules.

EXTERNAL Clause

The EXTERNAL clause allows an file to be shared between programs in the same execution instance.

General Format

```
IS EXTERNAL
```

General Rules

1. A file declared IS EXTERNAL shares its OPEN, and CLOSE state, READ/WRITE buffers, and file pointer state between separately compiled programs.
2. All programs using the file must have the same SELECT and FD declarations for the file, and all FD declarations must contain the IS EXTERNAL phrase.
3. A file description or data item may be declared IS EXTERNAL and IS GLOBAL simultaneously.

GLOBAL Clause

The GLOBAL clause

```
    IS GLOBAL
```

General Rules

1. A file declared IS GLOBAL shares its OPEN, and CLOSE state, READ/WRITE buffers, and file pointer state between a program and the nested programs it contains, and which together form a single compilation unit.
2. The GLOBAL clause indicates that a data item, and any data items or conditions or indexes subordinate to it, may be accessed by any of the programs in a compilation unit.
3. The GLOBAL clause may only be used with data items having an 01 data level.
4. The GLOBAL clause may be used in the File Section, Working-Storage Section, Local Storage Section, and Linkage Section.
5. A file description or data item may be declared IS EXTERNAL and IS GLOBAL simultaneously.

LABEL RECORDS Clause

The LABEL RECORDS clause is recognized, and syntax is validated. However, the LABEL RECORDS clause is otherwise treated as commentary.

General Format

```
    [ LABEL RECORDS [ARE] {STANDARD} ]
                          {OMITTED }
```

General Rules

There are no General Rules.

LINAGE Clause

The LINAGE clause causes an internal counter to be maintained by print files which allows for the trapping of the END-OF-PAGE condition.

General Format

```
[ LINAGE IS page-lines LINES
     [ WITH FOOTING AT footing-line-number ]
     [ LINES AT TOP number-top-lines ]
     [ LINES AT BOTTOM number-bottom-lines ] ] .
```

Syntax

1. page-lines is a numeric data item, or integer literal
2. footing-line-number is a numeric data item, or integer literal
3. number-top-lines is a numeric data item or integer literal
4. number-bottom-lines is a numeric data item or integer literal

General Rules

1. page-lines is maintained as an internal counter. When counter reaches page-lines, the end-of-page condition is triggered, and can be trapped by the WRITE statement.
2. The LINES AT TOP number-top-lines clause is the number of line feeds that are written after advancing a page.
3. The LINES AT BOTTOM number-bottom-ines clause is the number of line feeds that are written before advancing a page.
4. The FOOTING AT footing-ine-number describes an area of the report above the LINES at BOTTOM, on which a page-footing can be printed.

RECORD Clause

The RECORD clause describes the size of the record.

Format 1

```
[RECORD [ CONTAINS min-int-1 [TO max-int-1                ]  CHARACTERS ]
```

Format 2

```
[RECORD [ IS VARYING IN SIZE [FROM min-int-2][TO max-int-2]] CHARACTERS ]
        [ DEPENDING ON record-depending ]
```

Syntax

1. Min-int-1 is an integer
2. Max-int-1 is an integer
3. Min-int-2 is an integer
4. Max-int-2 is an integer
5. record-depending is a data item described in the working-storage or linkage section.

General Rules

1. When used with LINE SEQUENTIAL files, the RECORD clause is treated as commentary.
2. The Format 1 RECORD clause can be used for fixed length records or variable length records.
3. If the Format 1 CONTAINS phrase includes declarations of min-int and max-int

CHARACTERS, and max-int is greater than min-int, then the records have variable length.

4. If the Format 1 CONTAINS phrase includes a single declaration of max-int CHARACTERS, or declarations of min-int and max-int CHARACTERS, where min-int is equal to max-int, then the records have fixed length.
5. The Format 2 RECORD clause is used for variable length records which may, optionally, rely on a DEPENDING ON phrase.
6. If a DEPENDING ON phrase is used, then the number of characters associated with the record length is derived from the value of the **record-depending** variable.
7. In the absence of a RECORD clause, the compiler computes whether the records in a file are of fixed or variable length, what the length of the record is, and if they are of variable length, what the minimum and maximum record lengths are.
8. If the record size, as calculated by the compiler, is outside of the range described by the RECORD clause, the compiler will report an error, and the abort the compilation. In the example below, a RECORD CONTAINS 20 CHARACTERS clause was entered in an FD with a record that contained 30 characters:
reswords.fd:5: Error: Record size too large 'reswords-record' (30)

RECORDING MODE Clause

The RECORDING MODE clause is is recognized, and syntax is validated. RECORDING MODE F and RECORDING MODE V are supported. RECORDING MODE U, S, are treated as commentary.

General Format

```
[ RECORDING MODE IS { F } ]
                    { V }
                    { U }
                    { S }
```

General Rules

1. LINE SEQUENTIAL files declared with RECORDING MODE F are stored with fixed record length. Trailing spaces are not removed.
2. LINE SEQUENTIAL files declared with RECORDING MODE V are stored with variable record length.

REPORT IS Clause

The REPORT IS clause is recognized, and syntax is validated. However, the REPORT IS clause is otherwise treated as commentary.

General Format

```
[ { REPORT IS   } report-name ]
  { REPORTS ARE }
```

General Rules

There are no General Rules.

VALUE OF Clause

The VALUE OF clause is recognized, and syntax is validated. The handling of the VALUE OF clause is affected by:

-fvalue-of-id-priority or

value-of-id-priority:yes

With either of these conditions, the literal or data element that is the target of the VALUE OF FILE-ID clause in the FD overrides the target of the ASSIGN clause for the file. Otherwise, this setting is treated as commentary.

General Format

```
[ VALUE OF { LABEL   } IS value-of-name ]
           { FILE-ID }
```

Syntax

1. Value-of-name is a literal or data element. When associated with the VALUE OF FILE-ID phrase, it refers to the file's name, as it would appear in an ASSIGN TO clause.

General Rules

There are no General Rules.

4.4 WORKING-STORAGE SECTION.

The Working-Storage Section contains descriptions of data items used by the program internally.

General Format

```
[ WORKING-STORAGE SECTION. ]
 [ ws-data-level ws-data-description ] …
```

Syntax

The clauses supported by the data description are described in Paragraph 4.8

General Rules

1. ws-data-level is a data level number between 01 and 49 (inclusive), 66, 77, 78, or 88.

For more information about data level numbers, see 3.1 Data Level Numbers.

2. ws-data-description describes the format and size of a data item that is used by the program internally.
3. The default data initialization behaviour of the runtime corresponds to the declaration of the program in the PROGRAM-ID clause in the IDENTIFICATION DIVISION. If the program is described as IS INITIAL PROGRAM, IS COMMON PROGRAM, or IS RECURSIVE PROGRAM, the default behaviour of the runtime is to initialize alphanumeric data items to spaces, numeric data items to zeroes, and pointer data items to null every time the program is loaded.
4. If there is no specific description, the program is considered a RESIDENT program, and the default behaviour of the runtime is to initialize alphanumeric data items to spaces, numeric data items to zeroes, and pointer data items to null the first time the program is loaded. When the program is called subsequently, the data items in the Working-Storage Section retain their state, unless the program has been the target of the CANCEL statement.
5. Working-Storage variables may be created with an initial value using the VALUE clause.
6. The default data initialization behaviour of the runtime can be altered with the default-byte setting and use-defaultbyte settings in the compiler configuration file.
   For example, to cause all Working-Storage data items except those with explicit VALUE statements to be initialized to low-values when a program is initially loaded:
   In the compiler configuration flag, set :
   Defaultbyte: 0
   Use-defaultbyte: yes

4.5 LOCAL-STORAGE SECTION.

The Local-Storage Section contains descriptions of data items used by nested programs.

General Format

```
[ LOCAL-STORAGE SECTION. ]
  [ ls-data-level ls-data-description ] …
```

Syntax

The clauses supported by the data description are described in Paragraph 4.8

General Rules

1. ls-data-level is a data level number between 01 and 49 (inclusive), 66, 77, 78, or 88. For more information about data level numbers, see 3.1 Data Level Numbers.
2. ls-data-description describes the format and size of a data item that is used by the nested program.
3. The general rules for the Local Storage Section are the same as the General Rules for the Working-Storage Section except that the data items in Local Storage Section are always initialized when the program is CALL'ed.
4. LOCAL-STORAGE is allocated in one contiguous block of memory. (Prior to version 3.8, each 01 level was allocated as a separated block)

4.6 LINKAGE SECTION.

The Linkage Section contains descriptions of data items that have been passed to a CALL'ed subprogram.

General Format

```
[ LINKAGE SECTION. ]
  [ lk-data-level lk-data-description ] …
```

Syntax

The clauses supported by the data description are described in Paragraph 4.8

General Rules

1. ls-data-level is a data level number between 01 and 49 (inclusive), 66, 77, 78, or 88. For more information about data level numbers, see paragraph 3.1 Data Level Numbers.
2. ls-data-description describes the format and size of a data item that has been passed to a CALL'ed subprogram. For more information on the CALL statement, see paragraph 4.3.6 CALL Statement.
3. Data items declared in the Linkage Section are named in the USING clause of the PROCEDURE DIVISION. For more information on the USING clause, see paragraph 4.1.1 USING Clause.
4. Data items declared in the Linkage Section take their initial value from the CALL'ing program. In the cases where the variables are passed BY REFERENCE, their value is returned to the CALL'ing program after processing in the subprogram, and may be changed. For more information on the BY clause see paragraph 4.1.2 BY Clause.
5. The usage of a redefined field int the Linkage Section is allowed. The redefined field may subsequently be referenced in the PROCEDURE DIVISION USING clause.

Code Sample:

```
       LINKAGE  SECTION.
```

```
       01 X              PIC X(4) VALUE 'ABCD'.
       01 G              REDEFINES X.
         02 A            PIC X(2).
         02 B            PIC X(2).
        …
       PROCEDURE DIVISION  USING G.
        …
```

4.7 REPORT SECTION.

For a full description of the support of the Report Section, see the CitRW® Reference Manual.

4.8 DATA DESCRIPTION.

A data description describes a data item.

General Format

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

     [ REDEFINES identifier-1 ]

     [ IS EXTERNAL ]
     [ IS GLOBAL ]
     [ IS TYPEDEF ]

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

     [ [ USAGE IS ] usage-type ]

     [ [ SIGN IS ] {LEADING } [ SEPARATE CHARACTER ] ]
                   {TRAILING}

     [ OCCURS { integer TIMES }
              { int-1 TO int-2 TIMES DEPENDING ON reference }
        [ {ASCENDING } KEY IS {key-name} ... ] ...
          {DESCENDING}
        [ INDEXED BY {occurs-index} ... ] ]


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

     [ {JUSTIFIED} RIGHT ]
       {JUST}

     [ BLANK WHEN ZERO ]

     [ BASED ]
```

```
[ ANY LENGTH ]

[ RENAMES qualified-word [ {THRU    } qualified-word ]
                           {THROUGH}

[ { VALUE IS }   {value-lit                          } ]
  { VALUES ARE } { literal-1 [ {THRU    } literal-2 ] } ...
                               {THROUGH}
  [ WHEN SET TO FALSE literal-3 ]
```

Syntax

For syntax rules associated with a clause, see the clause description below:

4.8.1 ANY LENGTH Clause

General Format

```
[ ANY LENGTH ]
```

General Rules

1. Data items described as ANY LENGTH are not assigned a length at compile time.
2. Data items described as ANY LENGTH must be described as 01-level data items in the Linkage Section, and their PICTURE clause must have a single A, X, or 9 symbol.

Example:

LINKAGE SECTION.

01 param-1 PIC X ANY LENGTH.

4.8.2 BASED Clause

General Format

```
[ BASED ]
```

General Rules

1. Data items described as BASED are not assigned storage at compile time.

2. The ALLOCATE statement is used to allocate storage, and initialize data items described as BASED. For more information about the ALLOCATE statememt, see paragraph 5.3.4 ALLOCATE Statement.
3. The FREE statement is used to free storage assigned to data items described as BASED. For more information about the FREE statement, see paragraph 5.3.19 FREE Statement.

4.8.3 BLANK WHEN ZERO Clause

General Format

```
[ BLANK WHEN ZERO ]
```

General Rules

1. The BLANK WHEN ZERO clause may only be applied to data items that are NUMERIC or NUMERIC EDITED.
2. The effect of the BLANK WHEN ZERO clause is to store spaces in the data item when the value of the data item is zero.
3. Data items with a PICTURE category that is NUMERIC which contain the BLANK WHEN ZERO clause are considered to be NUMERIC EDITED.

4.8.4 EXTERNAL Clause

General Format

```
[ IS EXTERNAL ]
```

General Rules

1. When a file is declared as EXTERNAL, if the file-name is indicated as a variable name, in an ASSIGN DYNAMIC [file-var] clause, then file-var should be declared as EXTERNAL.
2. Variables declared as EXTERNAL must be declared as level 01 or level 77.
3. If [file-var] is not declared as EXTERNAL, then the default behavior of the COBOL-IT Compiler is to implicitly declare an external variable name, and assign it a name derived from the FD named in the SELECT clause.
4. The convention used is as follows:
   Consider the statement:
   SELECT myfile ASSIGN DYNAMIC file-var..…
   77 file-var pic x(8) value “customer”.
   In this case, file-var is not declared as EXTERNAL, so COBOL-IT issues the following warning:

```
'file-var' declared implicitly EXTERNAL AS
'FE_file_myfile_ASSIGN' (-fno-file-auto-external to disable).
```

Creating the implicit name based on the file name guarantees that the programmer will be able to give different names to [file-var] in different programs, and that they will nonetheless share the same value.

As noted in the Warning message, this functionality may be disabled using the compiler flag '-fno-file-auto-external'. However, when disabling this functionality, be aware, that if you have separate programs sharing the same EXTERNAL file that also have file-var fields, then changes made between the programs will not automatically be shared. If file-var is declared explicitly as EXTERNAL, then this condition does not apply.

5. A file declared EXTERNAL shares its OPEN, and CLOSE state, READ/WRITE buffers, and file pointer state between separately compiled programs within the run unit.
6. All programs using the file must have the same SELECT and FD declarations for the file, and the same variable describing the name of the file as EXTERNAL.
7. The VALUE clause can be applied to fields described as EXTERNAL. When a VALUE CLAUSE is applied to a field described as EXTERNAL, the runtime looks in memory for the external symbol. If the symbol is found, having been allocated by a previous run or CALL'ing program, then the VALUE clause is ignored. If the symbol is not found, then the runtime allocates the variable, and fills the allocation with the VALUE clause. In summary, the VALUE clause can only be applied in the first program allocating the field. When applied in programs other than the first program allocating the field, the VALUE clause is ignored.
8. The –ffile-auto-external compiler flag affects the way that the compiler treats variables describing file-names for files described as EXTERNAL.

4.8.5 GLOBAL Clause

General Format

```
[ IS GLOBAL ]
```

General Rules

1. The GLOBAL clause indicates that a data item, and any data items or conditions or indexes subordinate to it, may be accessed by any of the programs in a compilation unit.
2. The GLOBAL clause may only be used with data items having an 01 data level.
3. The GLOBAL clause may be used in the File Section, Working-Storage Section, Local Storage Section, and Linkage Section.

4.8.6 JUSTIFIED Clause

The JUSTIFIED RIGHT clause causes a MOVE of data to an ALPHABETIC or ALPHANUMERIC data item to align at the right-most character position of the data element.

General Format

```
[ {JUSTIFIED} RIGHT ]
  {JUST}
```

General Rules

1. JUST and JUSTIFIED are synonyms.
2. The JUSTIFIED clause can only be applied at the elementary data item level.
3. The JUSTIFIED RIGHT clause can only be used on ALPHABETIC and ALPHANUMERIC data items.
4. The JUSTIFIED clause alters the standard alignment rules in the following manner:
   Data is aligned at the right-most character position in the data element.
   If the value of the sending item is smaller than the maximum value expressible in the receiving item, and there are extra character spaces, the left-most character positions are space-filled. If there are not enough character places in the receiving data item to accommodate the value of the sending item, truncation takes place after the left-most character space has been filled.
5. The JUSTIFIED clause is not applied to the initial settings of variables with the VALUE clause.

4.8.7 LIKE Clause

The LIKE clause allows you to define the PICTURE, USAGE, and SIGN characteristics of a data item by copying them from a previously defined data item. The LIKE Statement can be used to describe elementary data items at level 01 through level 49.

General Format

```
LIKE data-name
```

Syntax

1. data-name can be an elementary, or group item.

Code Sample:

```
01 field-1 PIC X(10).
01 field-2  LIKE field-1.
```

```
*>In the example above,  field-2 is described with the definition PIC X(10).

01 RECORD1.
   05 element-1   PIC X(10).
   05 element-2   PIC 9(4).

01 RECORD2 LIKE RECORD1.
*> In the example above, RECORD2 is described with the definition :
01 RECORD2.
   05 element-1   PIC X(10).
   05 element-2   PIC 9(4).
```

4.8.8 OCCURS Clause

General Format

```
[ OCCURS { integer TIMES }
         { int-1 TO int-2 TIMES DEPENDING ON identifier-1 }
   [ {ASCENDING } KEY IS {key-name} ... ] ...
     {DESCENDING}
   [ INDEXED BY {occurs-index} ... ] ]
```

Syntax

1. integer is a positive integer describing the size of the table.
2. int-1 is an integer describing the minimum number of occurrences.
3. int-2 is an integer describing the maximum number of occurrences. int-2 must be greater than int-1.
4. identifier-1 is a numeric data item.
5. key-name is a data name contained within the occurs clause.
6. occurs-index is an index name.

General Rules

1. The OCCURS clause describes the number of occurrences in a table.
2. A fixed-length table is described with the *OCCURS integer TIMES* clause.
3. A variable-length table is described with the *OCCURS int-1 TO int-2 TIMES DEPENDING ON identifier-1* clause.
4. When describing a variable-length table, with the number of occurrences *DEPENDING ON identifier-1*, *identifier-1* must be a numeric data item, and must have a value greater than or equal the minimum number of occurrences described by int-1, and less than or equal the maximum number of occurrences described by int-2.
5. If the table being described contains data that is sorted, then including this information in the description of the table using the ASCENDING KEY, DESCENDING KEY, and INDEX clauses allows for the use of the SEARCH ALL statement on the contents of the table.
6. When using the *ASCENDING KEY IS key-name* clause, key-name must be a data item described in the table, and the data in the table must be sorted on key-name in ASCENDING order.

7. When using the *DESCENDING KEY IS key-name* clause, key-name must be a data item described in the table, and the data in the table must be sorted on key-name in DESCENDING order.
8. When multiple instances of ASCENDING KEY and DESCENDING KEY are described, the first instance is considered the primary sort key, the second instance the secondary sort key, and so forth. The sort keys are applied in the order in which they are listed.
9. The data item referenced in the *INDEXED BY index* clause is a data name described as USAGE INDEX. For more information about USAGE INDEX data items, see USAGE Clause.
10. The elements subordinate to the OCCURS clause must be referred to in the program with subscripts.
11. The default behaviour of the compiler is to require that phrases with OCCURS clause not be used at the 01-level. To alter this behaviour, modify the default.conf file, as follows:
```
top-level-occurs-clause:      ok
```

4.8.9 PICTURE Clause

General Format

```
[ {PICTURE} IS picture-string ]
  {PIC }
```

Syntax

1. picture-string uses symbols, with notations to indicate repeat symbols, to describe the characteristics of each of the character positions of a data item.
2. Inside picture-string, the following picture symbols are supported, in proper context:

General Rules

The General Rules for the PICTURE clause are described in 4.8.10 Picture Symbols, and 4.8.11 Picture Data Categories.

4.8.10 Picture Symbols

Classifying Character Types

PIC	Character Type	General Rules
1	Boolean character	Each symbol '1' represents the position of a Boolean character. Each symbol '1' adds 1 byte to the size of the item. PIC 1 may be used with USAGE DISPLAY, COMP, COMP-5, COMP-X, BIT. When used with USAGE COMP, COMP-5, COMP-X the minimum

		number of bytes needed to store the required number of bits is allocated. When used with USAGE DISPLAY, one byte per bit is allocated. Each byte contains “0” or “1”. When using USAGE BIT, the exact numbers of bits are used that are allocated in the current bit field. For example, the following declaration allocates 3 bytes: 01 bx PIC 1(24) USAGE COMP-5. The maximum number of bits allowed is 64. Display of a PIC 1 field will display the bits, for example, “000101”. PIC 1 fields are numeric and may be used in computations and MOVEs for their integer value. Literary binary constants are now supported, for example VALUE B’1011’. Data items described as PIC 1 USAGE BIT are allowed to have an OCCURS clause. For example: 05 element-1 PIC 1 USAGE BIT OCCURS 10000. Data described as USAGE BIT may be initialized at program startup, as indicated by the `initialize-filler:yes` compiler configuration flag, and as indicated by the VALUE clause. As an example, the FILLER in the group ALL-FLAGS below is initialized to B”1” when initialize-filler:yes is set in the compiler configuration file : 01 ALL-FLAGS. 03 FILLER PIC 1 BIT 88 ALL-DONE VALUE B”1” FALSE B”0”. … INITIALIZE ALL-FLAGS. …
9	Numeric digit	Each symbol '9' represents the position of a numeric digit. For USAGE DISPLAY, each symbol ‘9’ represents a numeric digit, values 0 through 9, and each symbol ‘9’ adds 1 byte to the size of the item. For other USAGE cases, correlation between the number of ‘9’s and the size of the item is described in the General Rules of the USAGE clause.

A	Alphabetic character	Each symbol 'A' represents the position of an alphabetic character. Each symbol 'A' adds 1 byte to the size of the item.
N	National character	Each symbol 'N' represents the position of a national character. Each symbol 'N' adds 1 byte to the size of the item.
S	Indicates an operational sign	The symbol 'S' indicates that internal data storage will include information about the operational sign. For USAGE DISPLAY, each symbol 'S' adds 1 byte to the size of the item when the SIGN SEPARATE clause is present. For other USAGE cases, the implementation of the 'S' in data storage is described in the General Rules of the USAGE clause.
X	Alphanumeric character	Each symbol 'X' represents the position of an alphanumeric character. Each symbol 'X' adds 1 byte to the size of the item.
Replacement Editing Characters		
*	Leading digit replaced by * when 0	Each symbol '*' represents the position of a numeric character, and further represents that leading 0's will be replaced with the "*" character. Each symbol '*' adds 1 byte to the size of the item.
Z	Leading digit replaced by space when 0	Each symbol 'Z' represents the position of a numeric character, and further represents that leading 0's will be replaced with the space character. Each symbol 'Z' adds 1 byte to the size of the item.
Positional Insertion Editing Characters		
,	Position where a comma is inserted	Each symbol ',' (comma) represents the position of the insertion of a "," (comma). Each symbol ',' adds 1 byte to the size of the item. Note- When the DECIMAL-POINT IS COMMA clause is used in a program, the COMMA editing character is interpreted as a PERIOD, and the PERIOD editing character is interpreted as a COMMA.
.	Position where a decimal point is inserted	The symbol '.' (period) represents the position of the insertion of a "." (period). Each symbol ',' adds 1 byte to the size of the item.

		Note- When the DECIMAL-POINT IS COMMA clause is used in a program, the PERIOD editing character is interpreted as a COMMA, and the COMMA editing character is interpreted as a PERIOD.
0	Position where a zero is inserted	The symbol '0' represents the position of the insertion of a "0" (zero). Each symbol '0' adds 1 byte to the size of the item.
B	Position where a space is inserted	The symbol 'B' represents the position of the insertion of a " " (space). Each symbol 'B' adds 1 byte to the size of the item.
/	Position where a / is inserted	The symbol '/' represents the position of the insertion of a "/" (slash). Each symbol '/' adds 1 byte to the size of the item.
+	Sign control symbol	The symbol '+' represents the position of the insertion of a "+" (plus sign). Each symbol '+' adds 1 byte to the size of the item.
-	Sign control symbol	The symbol '-' represents the position of the insertion of a "-" (mins sign). Each symbol '-' adds 1 byte to the size of the item.
C CR	Sign control symbol	The symbol 'C' or 'CR' represents the position of the insertion of a "C" or "CR" (CREDIT). The symbols 'C' or 'CR' must be placed at the end of the PICTURE string. "C" adds 1 byte to the size of the item. "CR" adds 2 bytes to the size of the item.
D DB	Sign control symbol	The symbol 'D' or 'DB' represents the position of the insertion of a "D" or "DB" (DEBIT). The symbols 'D' or 'DB' must be placed at the end of the PICTURE string. "D" adds 1 byte to the size of the item. "DB" adds 2 bytes to the size of the item.
$	Position where currency symbol is inserted	**3.** The symbol '$' represents the position of the insertion of the currency symbol. The currency symbol is either the '$' symbol, or by the currency symbol named in the CURRENCY clause in the SPECIAL-NAMES paragraph. **4.** "$" adds 1 byte to the size of the item.
Positioning Scaling Positions		
P	Scales a number by a power of 10	The 'P' symbol represents the position of a multiplication by a power of 10. Thus, PIC 9PP represents a number that is stored internally as

		a single digit (0-9), but represents the numbers 000, 100, 200, … 900. Similarly, PIC PP9 represents a number that is stored internally as a single digit (0-9), but represents the decimal numbers .000, .001, .002… .009. The symbol ‘P’ can only appear in either the left-most or right-most positions of a PICTURE string. An assumed decimal point, marked by the “V” symbol, may be used either to the right of the high-order ‘P’ symbol(s) or to the left of the low-order ‘P’ symbols. The “.” Symbol may not be used in the same picture string as the ‘P’ symbol. When included in a MOVE to a numeric field, or a numeric comparison, or a calculation, the data item is represented as though each ‘P’ symbol were a place-holder containing a 0.
Positioning Assumed Decimal Point		
V	Position of an assumed decimal point	The symbol 'V' represents the position of an assumed decimal point. The symbol 'V' does not add to the size of the item.

4.8.11 Picture Data Categories

1. A PICTURE clause defines the subject of the entry to fall into one of the following categories of data:
    a. Alphabetic
    b. Alphanumeric
    c. Alphanumeric Edited
    d. Boolean
    e. National
    f. National Edited
    g. Numeric
    h. Numeric Edited

The descriptions of each of these categories follows:

Category	General Rules
Alphabetic	2. An Alphabetic Picture String contains

	"A" symbols only.
Alphanumeric	An alphanumeric picture string must have either an "X" symbol, or some combination of "X", "A", and "9" symbols. A picture string consisting of only "A" symbols or only "9" symbols is not alphanumeric.
Alphanumeric Edited	An alphanumeric-edited picture string must have one or more "X" symbols, one or more "A" symbols, and at least one of the insertion symbols "B", "0", or "/".
Boolean	A boolean picture string must have one or more "I" symbols.
National	A national picture string must have one or more "N" symbols.
National Edited	A national-edited picture string must have one or more "N" symbols , and at least one of the insertion symbols "B", "0", or "/".
Numeric	A numeric picture string must have one or more "9" symbols, and can also contain one or more "P", "S", or "V" symbols. When signed, numeric picture strings can contain the "+" or "-" symbol.
Numeric Edited	A numeric edited picture string contains some combination of one or more of the symbols "9" , "*","Z", comma, period, "0", "B", "/","+", "-","CR", "DB", "$","P","V"

4.8.12 Rules for Alignment of Data

General Rules

1. When data is moved into a data element, the rules for alignment of data vary according to the data category of the data element receiving the data.
    a. For numeric data elements with a fixed decimal point:
        i. Data is aligned on the fixed decimal point.
        ii. Digits to the left and right of the decimal point are moved from the decimal point to the left for integer digits, and from the decimal point to to the right for decimal digits.
        iii. If there are not enough decimal places in the receiving data item to accommodate the data to the left of the decimal point, truncation of the high-order bytes takes place.
        iv. If there are not enough decimal places in the receiving data item to accommodate the data to the right of the decimal point, truncation of the low-

order bytes takes place.

v. Extra decimal places in the receiving data item to the left or the right of the decimal point are zero-filled.
vi. If no decimal point is indicated explicitly, then numeric data items are assigned an implicit decimal point after the right-most integer digit.
vii. Boolean data elements are treated as numeric data elements with an integer value, and with an implicit decimal point after the right-most integer digit.

b. For numeric edited data elements with a fixed decimal point:
   i. Data alignment rules are the same as for numeric data elements with a fixed decimal point, except that numeric edited data elements may describe replacement characters for leading zeroes.
c. For alphabetic, alphanumeric, alphanumeric-edited, national, national-edited data elements:
   i. Data is aligned at the left-most character position in the data element.
   ii. Extra character places in the receiving data item are space-filled.
   iii. If there are not enough character places in the receiving data item to accommodate the data being sent, truncation takes place after the right-most character space is filled.
   iv. Data alignment rules may be altered if the receiving data item has a JUSTIFIED clause. Alignment rules for the JUSTIFIED clause are described in 4.8.6 JUSTIFIED clause.

4.8.13 REDEFINES Clause

The REDEFINES clause causes two data items with different names, and potentially different data descriptions to occupy the same internal memory storage.

General Format

```
[ REDEFINES identifier-1 ]
```

Syntax

1. identifier-1 is a data item immediately preceding the redefining data item.

General Rules

1. The data item that REDEFINES identifier-1 must immediately follow identifier-1.
2. The data item that REDEFINES identifier-1 must have the same level number as identifier-1. The level number may not be 66, 78, or 88.
3. If the data item that REDEFINES identifier-1 is smaller than identifier-1, it will store data that begins with the first byte of identifier-1, and continues for its full length. The truncation is in the right-most bytes of identifier-1.
4. If the data item that REDEFINES identifier-1 is larger than identifier-1, there is potential for a memory access violation as the memory that is defined is only as long as identifier-1.

5. The data item that REDEFINES identifier-1 may not contain any VALUE clauses, other than those contained in level-88 condition statements.

4.8.14 RENAMES Clause

General Format

```
[ RENAMES identifier-1 [ {THRU    } identifier-2 ] .
                         {THROUGH}
```

Syntax

1. identifier-1 is a data element
2. identifier-2 is a data element declared after identifier-1. All data elements between identifier-1 and identifier-2 are grouped together under the original data descriptor in the clause.

General Rules

1. The RENAMES clause is only used with level-66 data items.
2. Level-66 data items may only rename items with data levels between 02 and 49 (inclusive). Level-66 data items may not rename level-01 data items, or level-66, level-77, level-78, or level-88 data items.
3. The data item that RENAMES identifier-1 (thru identifier-2) must immediately follow identifier-1 (thru identifier-2).
4. None of the data items in the renamed data item(s) may have variable length.
5. When the THRU clause is used, the level-66 data item provides a new name for all contiguous data storage beginning at identifier-1, up to and including identifier-2. Identifier-2 must be declared after identifier-1.
6. When the THRU clause is used, the level-66 data item is treated as a group item that contains all of the contiguous data elements from identifier-1 through identifier-2.
7. THRU and THROUGH are synonyms.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  rename1.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 customer-data.
         03 customer-name          PIC X(25).
         03 customer-street-number PIC X(10).
         03 customer-street        PIC X(25).
       66 customer-street-addr RENAMES
            customer-street-number THRU customer-street.
       77 ws-dummy pic x.
       PROCEDURE DIVISION.
```

```
       0000-MAIN.
           MOVE "John Doe" to customer-name.
           MOVE "25" to customer-street-number.
           MOVE "Main Street" to customer-street.

           DISPLAY customer-street-addr line 10 col 10.
           ACCEPT ws-dummy line 10 col 50.
           STOP RUN.
```

4.8.15 SIGN Clause

The SIGN clause determines how SIGN information is stored in USAGE DISPLAY data items that are described with the "S" symbol.

General Format

```
[ [ SIGN IS ] {LEADING } [ SEPARATE CHARACTER ] ]
              {TRAILING}
```

General Rules

1. The SIGN clause is only used with numeric data items that have a USAGE DISPLAY.
2. A SIGN clause may be described at a group level, in which case it will apply to all of the elementary items within that group.
3. If an elementary item within that group contains its own SIGN clause that is different than the SIGN clause of the group item, the SIGN clause of the elementary data item takes precedence over the SIGN clause of the group item.
4. When the SEPARATE CHARACTER phrase is used, the positive sign is stored as a "+" character, and the negative sign is stored as a "-" character.
5. When the SEPARATE CHARACTER phrase is **not** used, the negative SIGN is encoded, according to whether LEADING or TRAILING is used, as shown in the following table:

Digit	Sign Digit Character for:					
	Positively-signed values			Negatively-signed values		
	-fsign-ascii	-fsign-ebcdic	-fcarealia-sign	-fsign-ascii	-fsign-ebcdic	-fcarealia-sign
0	0(30)	{(7B)	0(30)	p(70)	}(7D)	0(30)
1	1(31)	A(41)	1(31)	q(71)	J(4A)	!(21)
2	2(32)	B(42)	2(32)	r(72)	K(4B)	"(22)
3	3(33)	C(43)	3(33)	s(73)	L(4C)	#(23)
4	4(34)	D(44)	4(34)	t(74)	M(4D)	$(24)
5	5(35)	E(45)	5(35)	u(75)	N(4E)	%(25)
6	6(36)	F(46)	6(36)	v(76)	O(4F)	&(26)

7	7(37)	G(47)	7(37)	w(77)	P(50)	'(27)
8	8(38)	H(48)	8(38)	x(78)	Q(51)	((28)
9	9(39)	I(49)	9(39)	y(79)	R(52)	)(29)

Examples (compiled with –fsign-ascii) :

Picture Clause	Storage
PIC S9(4) SIGN IS TRAILING VALUE -1230	123p
PIC S9(4) SIGN IS TRAILING VALUE -1231	123q
PIC S9(4) SIGN IS TRAILING VALUE -1232	123r
PIC S9(4) SIGN IS TRAILING VALUE 1230	1230
PIC S9(4) SIGN IS TRAILING VALUE 1231	1231
PIC S9(4) SIGN IS TRAILING VALUE 1232	1232
PIC S9(4) SIGN IS LEADING VALUE -0321	p321
PIC S9(4) SIGN IS LEADING VALUE -1321	q321
PIC S9(4) SIGN IS LEADING VALUE -2321	r321
PIC S9(4) SIGN IS LEADING VALUE 0321	0321
PIC S9(4) SIGN IS LEADING VALUE 1321	1321
PIC S9(4) SIGN IS LEADING VALUE 2321	2321

6. If there is no SIGN clause, the default is to assign the storage rules for the SIGN IS TRAILING clause.

Examples (compiled with –fsign-ascii) :

Picture Clause		Storage
PIC S9(4)	VALUE -1230	123p
PIC S9(4)	VALUE 1230	1230

4.8.16 SYNCHRONIZED Clause

The SYNCHRONIZED Clause affects the manner in which binary data is stored, with the intention of optimizing the performance.

General Format

```
[ {SYNCHRONIZED} [{LEFT } ] ]
```

```
        {SYNC}          [{RIGHT}]
```

General Rules

1. SYNC and SYNCHRONIZED are synonyms.
2. When a data item is SYNCHRONIZED, it cannot share any unused space within its natural boundaries with any other data item.
3. SYNCHRONIZED LEFT causes the data to be aligned to the left-most byte of the natural boundary in which it is stored. Unused space, if there is any, will appear in the right-most bytes of the natural boundary.
4. SYNCHRONIZED RIGHT causes the data to be aligned to the right-most byte of the natural boundary in which it is stored. Unused space, if there is any, will appear in the left-most bytes of the natural boundary.

4.8.17 TYPEDEF Clause

The TYPEDEF field attribute allows you to define a “user-defined” USAGE.

General Format

```
    [ IS TYPEDEF ]
```

Syntax

1. A data description with the IS TYPEDEF attribute must have a data level 01 or a data level 77.

General Rules

1. The TYPEDEF clause indicates that the data description is a typedef declaration, and not a variable declaration. The data name can then be used with the USAGE clause.
2. TYPEDEF declarations cannot store data.
3. If the data description with the TYPEDEF clause is a level-01 data item, all subordinate data names, including level-66, level-78, and level-88 data names, are part of the typedef declaration.
4. Subordinate data names must be qualified with the name of the data name using the USAGE [typedef] clause.
5. A data item with USAGE [TYPEDEF] may be the subject of an OCCURS clause.
   For example:
   05 cust-data usage customer-info occurs 1000 times.
6. TYPEDEFs have GLOBAL scope by default. This behaviour may be changed by use of the

–fno-global-typedef compiler flag.

Code Sample

```
      *
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  TYPEDEF1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 CUSTOMER-INFO IS TYPEDEF.
         05 CUSTOMER-ADDRESS.
           10 CUSTOMER-STREET-NUM  PIC X(5).
           10 CUSTOMER-STREEN-NAME PIC X(20).
           10 CUSTOMER-CITY        PIC X(15).
           10 CUSTOMER-STATE       PIC X(4).
           10 CUSTOMER-ZIP         PIC X(5).

       01 CUST-DATA USAGE CUSTOMER-INFO.

       77 WS-DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
           MOVE "1600" TO CUSTOMER-STREET-NUM OF CUST-DATA.
           MOVE "PENNSYLVANIA AVE." TO CUSTOMER-STREEN-NAME OF CUST-DATA.
           MOVE "WASHINGTON" TO CUSTOMER-CITY OF CUST-DATA.
           MOVE "DC" TO CUSTOMER-STATE OF CUST-DATA.
           MOVE "20202" TO CUSTOMER-ZIP OF CUST-DATA.

           DISPLAY CUSTOMER-ADDRESS OF CUST-DATA LINE 10 COL 10.
           DISPLAY "TYPEDEF1 FINISHED!" LINE 15 COL 10.
           ACCEPT WS-DUMMY LINE 15 COL 30.

           STOP RUN.
```

4.8.18 USAGE Clause

The USAGE clause affect the manner in which a data type is stored in memory.

General Format

```
     [ [ USAGE IS ] { usage-type           } ]
```

Syntax

usage-type may be any of the following :

```
{ BINARY                }
{ PACKED-DECIMAL        }
{ DISPLAY               }
```

```
{ INDEX                  }
{ POINTER                }
{ PROGRAM-POINTER        }
{ BINARY-C-LONG          }
{ BINARY-C-LONG SIGNED   }
{ BINARY-C-LONG UNSIGNED }
{ BINARY-CHAR            }
{ BINARY-CHAR SIGNED     }
{ BINARY-CHAR UNSIGNED   }
{ BINARY-DOUBLE          }
{ BINARY-DOUBLE SIGNED   }
{ BINARY-DOUBLE UNSIGNED }
{ BINARY-LONG            }
{ BINARY-LONG SIGNED     }
{ BINARY-LONG UNSIGNED   }
{ BINARY-SHORT           }
{ BINARY-SHORT SIGNED    }
{ BINARY-SHORT UNSIGNED  }
{ COMPUTATIONAL          }
{ COMP                   }
{ COMPUTATIONAL-1        }
{ COMP-1                 }
{ COMPUTATIONAL-2        }
{ COMP-2                 }
{ COMPUTATIONAL-3        }
{ COMP-3                 }
{ COMPUTATIONAL-4        }
{ COMP-4                 }
{ COMPUTATIONAL-5        }
{ COMP-5                 }
{ COMPUTATIONAL-6        }
{ COMP-6                 }
{ COMPUTATIONAL-X        }
{ COMP-X                 }
{ FLOAT-SHORT            }
{ FLOAT-LONG             }
{ SIGNED-INT             }
{ SIGNED-LONG            }
{ SIGNED-SHORT           }
{ UNSIGNED-INT           }
{ UNSIGNED-LONG          }
{ UNSIGNED-SHORT         }
{ user_typedef           }
```

Syntax

1. user_typedef is a field name defined as ‘IS TYPEDEF’.

USAGE Clause General Rules

1. If no USAGE clause is declared, then USAGE DISPLAY is implied.
2. If a USAGE clause is declared or implied on a group level data item, all subordinate data names inherit the USAGE clause declaration.
3. For General Specifications of all of the USAGE types, see Appendix 6.1 Data Memory

Allocation.

USAGE BINARY

Data Storage

Data Description	Data Storage
PIC S9(3) USAGE BINARY VALUE -123	FF 85
PIC 9(3) USAGE BINARY VALUE 123	00 7B

General Rules

1. BINARY data is sized according to the number of 9's in the PICTURE clause, and according to the setting of the "binary-size" compiler configuration flag. For details, see the documentation of the binary-size compiler configuration flag in 6**.3 Compiler Configuration File.**

USAGE PACKED-DECIMAL

Data Storage

Data Description	Data Storage
PIC S9(3) USAGE BINARY VALUE -123	12 3D
PIC S9(3) USAGE BINARY VALUE 123	12 3C

General Rules

1. USAGE PACKED-DECIMAL, USAGE COMP-3, and USAGE COMPUTATIONAL-3 are synonyms.
2. COMP-3/PACKED-DECIMAL data allocates 4 bits per "9" in ithe PICTURE clause, plus a trailing 4-bits for the sign. To calculate the size of a COMP-3/PACKED-DECIMAL data item, add 1 (for the sign) to the number of 9's in the PICTURE clause, divide by 2, and if the result is not a whole number, round it up to the next integer.
3. The half-byte D indicates a negative sign, the half-byte C indicates a positive sign..

USAGE DISPLAY

USAGE NATIONAL

For details onUSAGE DISPLAY and USAGE NATIONAL data storage, see 4.8.12 Rules for Alignment of Data and 4.8.15 Sign Clause.

General Rules

1. The USAGE DISPLAY clause indicates that a data item is Alphabetic, Alphanumeric, Alphanumeric Edited, Numeric, Numeric Edited, National or National Edited.
2. If no USAGE clause is declared, then USAGE DISPLAY is implied.

USAGE INDEX

Data Storage

Data Description	Data Storage
USAGE INDEX VALUE 0	00 00 00 00
USAGE INDEX VALUE 1	01 00 00 00

General Rules

1. The USAGE INDEX clause indicates that a data item is an index is being used in table-handling functions.
2. USAGE INDEX data allocates 32-bits in native format.
3. USAGE INDEX data does not allow the use of the PICTURE clause.
4. USAGE INDEX data does support negative values.

USAGE POINTER

USAGE PROCEDURE-POINTER

USAGE PROGRAM-POINTER

Data Storage

Data Description	Data Storage
USAGE POINTER (initial)	00 00 00 00
USAGE POINTER (set to address of data-item)	D0 41 11 6C
USAGE PROCEDURE-POINTER (initial val)	00 00 00 00
USAGE PROCEDURE-POINTER (set)	10 10 11 6C
USAGE PROGRAM-POINTER (initial value)	00 00 00 00
USAGE PROGRAM-POINTER (set)	00 10 11 6C

General Rules

1. Data items described with USAGE POINTER, USAGE PROCEDURE-POINTER, and USAGE PROGRAM-POINTER are initialized to NULL when a program is initially loaded into memory.
2. Data items described with USAGE POINTER, USAGE PROCEDURE-POINTER, and USAGE PROGRAM-POINTER hold unsigned 32-bit binary values on machines with 32-bit architectures, and unsigned 64-bit binary values on machines with 64-bit architectures.
3. Data items described with USAGE POINTER, USAGE PROCEDURE-POINTER, and USAGE PROGRAM-POINTER may be assigned a data address through the use of the ALLOCATE, or SET statement, and that memory may be released through the use of the FREE statement..

   Example:

   SET *pointer-var* TO ADDRESS OF *data-item*.

   SET *procedure-pointer-var* TO ENTRY "*entry-name-literal*".

   SET *program-pointer-var* TO ENTRY "*program-id-literal*".
4. The USAGE POINTER clause indicates that a data item is a data pointer, which can hold the address of a data item.
5. The USAGE PROGRAM-POINTER clause indicates that a data item is program pointer, which can hold the address of a program.
6. The program name from which the PROGRAM-POINTER is derived may be either the PROGRAM-ID of a program, or the entry-name of an ENTRY statement.
7. The USAGE PROCEDURE-POINTER clause indicates that a data item is procedure pointer, which can hold the address of a procedure.
8. The procedure name from which the PROCEDURE-POINTER is derived is the entry-name of an ENTRY statement.
9. USAGE POINTER, PROGRAM-POINTER, and PROCEDURE-POINTER do not allow the use of the PICTURE clause.

Code Sample

```
      *
       ...
       01 ERROR-PROC-FLAG     PIC X COMP-X VALUE 0.
       01 ERROR-PROC-ADDR     USAGE PROCEDURE-POINTER.
      *
           ...
           SET EXIT-PROC-ADDR TO ENTRY "EXIT-PROC".
           ...
           CALL "CBL_EXIT_PROC" USING EXIT-PROC-FLAG,
                                      EXIT-PROC-ADDR.
           STOP RUN.
           ...
       ENTRY "EXIT-PROC".
           DISPLAY "IN EXIT PROCEDURE".
           EXIT PROGRAM.
           STOP RUN.
      *
```

USAGE BINARY-C-LONG

BINARY-C-LONG SIGNED

BINARY-C-LONG UNSIGNED

Data Storage

Data Description	Data Storage
USAGE BINARY-C-LONG (initial)	00 00 00 00
USAGE BINARY-C-LONG VALUE 123.	7B 00 00 00
USAGE BINARY-C-LONG SIGNED (initial)	00 00 00 00
USAGE BINARY-C-LONG SIGNED VALUE -123.	85 FF FF FF
USAGE BINARY-C-LONG UNSIGNED (initial)	00 00 00 00
USAGE BINARY-C-LONG UNSIGNED VALUE 123.	7B 00 00 00

General Rules

1. Data items with USAGE BINARY-C-LONG are initialized to NULL when the program is loaded in memory.
2. BINARY-C-LONG data allocates the same amount of storage as does the “C” language “long” data type. Depending on the “C” compiler being used, this could be 32, or 64-bits.
3. BINARY-C-LONG data does not allow the use of the PICTURE clause.
4. BINARY-C-LONG data may be described as SIGNED or UNSIGNED.
5. BINARY C-LONG UNSIGNED data does not support negative values.
6. BINARY C-LONG data has the SIGNED attribute by default.

Code Sample

```
      *
       IDENTIFICATION DIVISION.
       PROGRAM-ID. FIBONACCI.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  IX                    BINARY-C-LONG VALUE 0.
       01  FIRST-NUMBER          BINARY-C-LONG VALUE 0.
       01  SECOND-NUMBER         BINARY-C-LONG VALUE 1.
       01  TEMP-NUMBER           BINARY-C-LONG VALUE 1.
       01  DISPLAY-NUMBER        PIC Z(19)9.
      *
       78  32-BIT-ITERATIONS     VALUE 46.
```

```
       78  64-BIT-ITERATIONS     VALUE 90.
       PROCEDURE DIVISION.
       MAIN.
           MOVE FIRST-NUMBER TO DISPLAY-NUMBER.
           DISPLAY DISPLAY-NUMBER.
           MOVE SECOND-NUMBER TO DISPLAY-NUMBER.
           DISPLAY DISPLAY-NUMBER.
           PERFORM VARYING IX FROM 1 BY 1 UNTIL IX = 32-BIT-ITERATIONS
               ADD FIRST-NUMBER TO SECOND-NUMBER GIVING TEMP-NUMBER
               MOVE SECOND-NUMBER TO FIRST-NUMBER
               MOVE TEMP-NUMBER TO SECOND-NUMBER
               MOVE TEMP-NUMBER TO DISPLAY-NUMBER
               DISPLAY DISPLAY-NUMBER
           END-PERFORM.

           STOP RUN.
```

USAGE BINARY-CHAR

BINARY-CHAR SIGNED

BINARY-CHAR UNSIGNED

Data Storage

Data Description	Data Storage
USAGE BINARY-CHAR (initial)	00
USAGE BINARY-CHAR VALUE 123.	7B
USAGE BINARY-CHAR SIGNED VALUE -123.	85
USAGE BINARY-CHAR UNSIGNED VALUE 123.	7B

General Rules

1. Data items with USAGE BINARY-CHAR are initialized to NULL when the program is loaded in memory.
2. BINARY-CHAR data designates a single byte using the native storage format.
3. BINARY-CHAR data does not allow the use of the PICTURE clause.
4. BINARY-CHAR data may be described as SIGNED or UNSIGNED.
5. BINARY CHAR UNSIGNED data does not support negative values.
6. BINARY CHAR data has the SIGNED attribute by default.

USAGE BINARY-DOUBLE,

BINARY-DOUBLE SIGNED

BINARY-DOUBLE UNSIGNED

Data Storage

Data Description	Data Storage
USAGE BINARY-DOUBLE (initial)	00 00 00 00 00 00 00 00
USAGE BINARY-DOUBLE VALUE 123.	7B 00 00 00 00 00 00 00
USAGE BINARY-DOUBLE SIGNED VALUE -123.	85 FF FF FF FF FF FF FF
USAGE BINARY-DOUBLE UNSIGNED VALUE 123.	7B 00 00 00 00 00 00 00

General Rules

1. Data items with USAGE BINARY-DOUBLE are initialized to NULL when the program is loaded in memory.
2. BINARY-DOUBLE data allocates a “traditional” double-word of storage (64-bits) using the native storage format.
3. BINARY-DOUBLE data does not allow the use of the PICTURE clause.
4. BINARY-DOUBLE data may be described as SIGNED or UNSIGNED.
5. BINARY-DOUBLE UNSIGNED data does not support negative values.
6. BINARY-DOUBLE data has the SIGNED attribute by default.

USAGE BINARY-LONG

BINARY-LONG SIGNED

BINARY-LONG UNSIGNED

Data Storage

Data Description	Data Storage
USAGE BINARY-LONG (initial)	00 00 00 00
USAGE BINARY-LONG VALUE 123.	7B 00 00 00
USAGE BINARY-LONG SIGNED VALUE -123.	85 FF FF FF
USAGE BINARY-LONG UNSIGNED VALUE 123.	7B 00 00 00

General Rules

1. Data items with USAGE BINARY-LONG are initialized to NULL when the program is

loaded in memory.
2. BINARY-LONG data allocates 32-bits of storage using the native storage format.
3. BINARY-LONG data does not allow the use of the PICTURE clause.
4. BINARY-LONG data may be described as SIGNED or UNSIGNED.
5. BINARY-LONG UNSIGNED data does not support negative values.
6. BINARY-LONG data has the SIGNED attribute by default.

USAGE BINARY-SHORT

BINARY-SHORT SIGNED

BINARY-SHORT UNSIGNED

Data Storage

Data Description	Data Storage
USAGE BINARY-SHORT (initial)	00 00
USAGE BINARY-SHORT VALUE 123.	7B 00
USAGE BINARY-SHORT SIGNED VALUE -123.	85 FF
USAGE BINARY-SHORT UNSIGNED VALUE 123.	7B 00

General Rules

1. Data items with USAGE BINARY-SHORT are initialized to NULL when the program is loaded in memory.
2. BINARY-SHORT data allocates 16-bits of storage using the native storage format.
3. BINARY-SHORT data does not allow the use of the PICTURE clause.
4. BINARY-SHORT data may be described as SIGNED or UNSIGNED.
5. BINARY-SHORT UNSIGNED data does not support negative values.
6. BINARY-SHORT data has the SIGNED attribute by default.

USAGE BIT

General Rules

1. USAGE BIT allocates a field as a bit field.
2. PIC 1(n) is the only picture allowed.
3. USAGE BIT may not redefine or be redefined.
4. Consecutive USAGE BIT fields are packed into the same byte area with a maximum size of 8 bytes (64 Bits).
5. If a USAGE BIT field does not fit in the current byte area because the resulting byte area would be greater than 64 bits, then a new byte area will be allocated. Remaining unused bits of a byte area are left unused.

As an example, a single byte would be allocated to contain these 3 fields b1, b2, and b3.

```
03 b1 PIC 1 USAGE BIT.
03 b2 PIC 1(3) USAGE BIT.
03 b3 PIC 1(4) USAGE BIT.
```

As an example, a single byte would be allocated to contain fields b4 and b5 in the example below, and 8 bytes would be allocated for b6:

```
03 b4 PIC 1 USAGE BIT.
03 b5 PIC 1(5) USAGE BIT.
03 b6 PIC 1(60) USAGE BIT.
```

6. The byte area stores bits in memory in "System Native" bit-order (like COMP-5).

Computational Data Types

USAGE COMPUTATIONAL

USAGE-COMP

Data Storage

Data Description	Data Storage
PIC S9(3) USAGE COMPUTATIONAL (initial)	00 00
PIC S9(3) USAGE COMPUTATIONAL VALUE -123.	FF 85
PIC S9(3) USAGE COMPUTATIONAL VALUE 123.	00 7B

General Rules

1. USAGE COMPUTATIONAL data is sized according to the number of 9's in the PICTURE clause, and according to the setting of the "binary-size" compiler configuration flag. For details, see the documentation of the binary-size compiler configuration flag in **6.3 Compiler Configuration File.**
2. USAGE COMPUTATIONAL and USAGE COMP are synonyms.

USAGE COMPUTATIONAL-1

USAGE COMP-1

Data Storage

Data Description	Data Storage
PIC S9(3) USAGE COMP-1 (initial)	00 00 00 00
PIC S9(3) USAGE COMP -1 VALUE -123.	00 00 F6 C2
PIC S9(3) USAGE COMP-1 VALUE 123.	00 00 F6 42

General Rules

1. USAGE COMPUTATIONAL-1 and USAGE COMP-1 are synonyms.
2. COMP-1 data allocates 32-bits of data, formatted as single-precision floating-point .
3. COMP-1 data does not allow the use of the PICTURE clause.
4. COMP-1 data does support negative values.

USAGE COMPUTATIONAL-2

USAGE COMP-2

Data Storage

Data Description	Data Storage
PIC S9(3) USAGE COMP-2 (initial)	00 00 00 00 00 00 00 00
PIC S9(3) USAGE COMP -2 VALUE -123.	00 00 00 00 00 C0 5E C0
PIC S9(3) USAGE COMP-2 VALUE 123.	00 00 00 00 00 C0 5E 40

General Rules

1. USAGE COMPUTATIONAL-2 and USAGE COMP-2 are synonyms.
2. COMP-2 data allocates 64-bits of data, formatted as double-precision floating-point . COMP-2 data does not allow the use of the PICTURE clause. COMP-2 data does support negative values.

USAGE COMPUTATIONAL-3

COMP-3

PACKED-DECIMAL

Data Storage

Data Description	Data Storage
PIC S9(3) USAGE COMP-3 (initial)	00 00
PIC S9(3) USAGE BINARY VALUE -123	12 3D
PIC S9(3) USAGE BINARY VALUE 123	12 3C

General Rules

1. USAGE PACKED-DECIMAL, USAGE COMP-3, and USAGE COMPUTATIONAL-3 are synonyms.
2. USAGE PACKED-DECIMAL data storage stores two decimal digits per byte, and includes a half-byte for the sign, which is placed in the right-most position.
3. The half-byte D indicates a negative sign, the half-byte C indicates a positive sign..
4. To calculate the SIZE of a USAGE PACKED-DECIMAL data item, add 1 to the number of decimal digits in the number (for the sign), divide by 2, and round the result up. Thus, for the number 255, there are three decimal digits- add 1 for the sign giving 4- and divide by 2, giving 2. The number 255, or -255 requires 2 bytes.

USAGE COMPUTATIONAL-4

COMP-4

Data Storage

Data Description	Data Storage
PIC S9(3) USAGE COMP-4 (initial)	00 00
PIC S9(3) USAGE COMP-4 VALUE -123.	FF 85
PIC S9(3) USAGE COMP-4 VALUE 123.	00 7B

General Rules

1. Data items with USAGE COMPUTATIONAL are initialized to xxxx when the program is

loaded in memory.
2. USAGE COMPUTATIONAL-4 and USAGE COMP-4 are synonyms.
3. COMP-4 data is sized according to the number of 9's in the PICTURE clause, and the setting of the "binary-size" compiler configuration flag or the setting of the pack-comp-4 compiler configuration flag. If pack-comp-4 is set to No (the default), then COMP-4 data follows the same rules as COMPUTATIONAL data with regards to sizing. If pack-comp-4 is set to Yes, then a binary-size setting of "1- - 8" is assumed for binary-size setting, and the sizing is done accordingly.

USAGE COMPUTATIONAL-5

COMP-5

Data Storage

Data Description	Data Storage
PIC S9(3) USAGE COMP-5 (initial)	00 00
PIC S9(3) USAGE COMP-5 VALUE -123.	85 FF
PIC S9(3) USAGE COMP-5 VALUE 123.	7B 00

General Rules

1. USAGE COMPUTATIONAL-5 and USAGE COMP-5 are synonyms.
2. COMP-5 data data is sized according to the number of 9's in the PICTURE clause, and according to the setting of the "binary-size" compiler configuration flag.
3. COMP-5 data is stored in native format.

USAGE COMPUTATIONAL-6

COMP-6

Data Storage

Data Description	Data Storage
PIC S9(3) USAGE COMP-6 (initial)	00 00
PIC S9(3) USAGE COMP-6 VALUE 123.	01 23

General Rules

1. USAGE COMPUTATIONAL-6 and USAGE COMP-6 are synonyms.
2. COMP-6 data allocates 4 bits per "9" in ithe PICTURE clause. To calculate the size of a

COMP-6 data item, divide the number of 9's in the PICTURE clause by 2, and if the result is not a whole number, round it up to the next integer. COMP-6 data does not allow negative values.

USAGE COMPUTATIONAL-X

COMP-X

Data Storage

Data Description	Data Storage
PIC 9(3) USAGE COMP-X (initial)	00 00
PIC 9(3) USAGE COMP-X VALUE -123.	FF 85
PIC 9(3) USAGE COMP-X VALUE 123.	00 7B

General Rules

1. COMP-X data is sized according to the number of 9's or the number of X's in the PICTURE clause.
2. When the PICTURE clause consists of 9's, COMP-X data assumes a binary-size setting of "1 - - 8" in calculating the number of bytes that correspond to the number of 9's in the PICTURE clause.
3. When the PICTURE clause consists of X's, one byte is assigned per X in the PICTURE clause.

USAGE FLOAT-SHORT

USAGE FLOAT-LONG

Data Storage

Data Description	Data Storage
PIC 9(3) USAGE FLOAT-SHORT (initial)	00 00 00 00
PIC 9(3) USAGE FLOAT-SHORT VALUE -123.	00 00 F6 C2
PIC S9(3) USAGE FLOAT-SHORT VALUE 123	00 00 F6 42
PIC 9(3) USAGE FLOAT-LONG (initial)	00 00 00 00 00 00 00 00
PIC 9(3) USAGE FLOAT-LONG VALUE -123.	00 00 00 00 00 C0 5E C0
PIC 9(3) USAGE FLOAT-LONG VALUE -123.	00 00 00 00 00 C0 5E 40

General Rules

1. USAGE FLOAT-SHORT is equivalent to USAGE COMP-1.
2. USAGE FLOAT-LONG is equivalent to USAGE COMP-2.

The "C" Data Types

1. The "C" data types are SIGNED-SHORT, UNSIGNED-SHORT, SIGNED-INT, UNSIGNED-INT, SIGNED-LONG, and UNSIGNED-LONG . They correspond to the "C" data types "short", "int", and "long".
2. The "C" data types do not allow a PICTURE clause.

USAGE SIGNED-SHORT

UNSIGNED-SHORT

USAGE SIGNED-INT

UNSIGNED-INT

USAGE SIGNED-LONG

UNSIGNED-LONG

Data Storage

Data Description	Data Storage
USAGE SIGNED-SHORT (initial)	00 00
USAGE UNSIGNED-SHORT VALUE -123.	85 FF
USAGE SIGNED-INT VALUE 123	7B 00
USAGE UNSIGNED-SHORT VALUE 123	7B 00
USAGE SIGNED-INT VALUE -123	85 FF FF FF
USAGE SIGNED-INT VALUE 123	7B 00 00 00
USAGE UNSIGNED-INT VALUE 123	7B 00 00 00
USAGE SIGNED-LONG VALUE -123.	85 FF FF FF FF FF FF FF
USAGE SIGNED-LONG VALUE 123	7B 00 00 00 00 00 00 00
USAGE UNSIGNED-LONG VALUE 123	7B 00 00 00 00 00 00 00

General Rules

1. SIGNED-SHORT and UNSIGNED-SHORT data allocate 16-bits of storage using the native storage format.
2. SIGNED-INT and UNSIGNED-INT data allocate 32-bits of storage using the native storage format.
3. SIGNED-LONG and UNSIGNED-LONG data allocate 64-bits of storage using the native storage format.
4. UNSIGNED-SHORT, UNSIGNED-LONG and UNSIGNED-INT data does not support negative values.

4.8.19 VALUE Clause

The VALUE clause can be used :

- To set the initial value of a data-item in the Working-Storage or Local Storage Section.
- To define a value or range of values associated with a level-88 condition.
- To set the value of a level-78 constant.

General Format

Format 1

The Format 1 VALUE clause is used to initialize data values when the program is initially loaded into memory.

```
[01-LEVEL TO 49-LEVEL, 77-LEVEL DATA-ITEM ]
        [ { VALUE IS }   {literal-1                    } ] .
```

Syntax

1. literal-1 is a numeric or alphanumeric literal which sets the initial value of the data-item in the Working-Storage Section or Local-Storage Section.

   Example:

   ```
   77 cobol-compiler          PIC X(8) VALUE “COBOL-IT”.
   77 nullterminated-var      PIC X(8) VALUE  Z”ABCDEFG”.
   77 current-year            PIC 9(4) VALUE 2011.
   ```

General Rules

1. Format-1 VALUE declarations are ignored in the FILE SECTION and LINKAGE SECTION.
2. Some API calls require that strings being passed to them be null-terminated. Prefixing a string with a Z has the effect of null-terminating the string.
3. If the parent data-item is numeric, literal-1 must be numeric. If the parent data-item is alphanumeric, literal-1 must be alphanumeric.
4. The VALUE declaration may not be applied to a data item with a variable size.

In a data item that is subordinate to an OCCURS clause, the VALUE clause is applied to every instance of the OCCURS.

5. When applied to a group item, the parent date item is considered to be alphanumeric, and the VALUE literal must be alphanumeric.
6. When a group-item contains a Format-1 VALUE clause, any VALUE statements on any of the subordinate data items will be ignored.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  VALUE1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
      *
       77 FILE-STATUS  PIC XX VALUE "10".

       77 WORK-HOURS PIC 99 VALUE 24.

       01 SAMPLE-TABLE.
         05 OCCURS 5 TIMES.
           10 SAMPLE-ELEMENT PIC X VALUE "Y".

       01 ADDRESS-1 VALUE "1600 PENNSYLVANIA AVE".
         05 ADDR-1 PIC X(5).
         05 ADDR-2 PIC X(13).
         05 ADDR-3 PIC X(3).

       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
           DISPLAY FILE-STATUS LINE 10 COL 10.
           DISPLAY WORK-HOURS LINE 11 COL 10.
           DISPLAY SAMPLE-ELEMENT(3) LINE 12 COL 10.
           DISPLAY ADDR-2 LINE 13 COL 10.

           DISPLAY "VALUE-1 FINISHED!" LINE 15 COL 10.
           ACCEPT DUMMY LINE 15 COL 30.
           STOP RUN.
```

Format 2

The Format 2 VALUE clause is used to define a value or range of values associated with a level-88 condition.

```
[ 88-LEVEL DATA-ITEM]
        [ { VALUE IS }   {literal-2                    } ] .
        { VALUES ARE } { literal-3 [ {THRU   } literal-4 ] } ...
                                   {THROUGH}
        [ WHEN SET TO FALSE literal-5 ]
```

Syntax

1. literal-2 is a numeric or alphanumeric literal which sets the value of the 88-level condition.

Examples:
77 file-status PIC XX.
88 end-of-file VALUE “10”.

77 starting-hour PIC 99.
88 afternoon-hours VALUES ARE 12 THRU 18.

77 decision-flag PIC X.
88 yes-decision VALUE “Y”
WHEN SET TO FALSE “N”.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  VALUE2.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
      *
       77 FILE-STATUS  PIC XX.
         88 END-OF-FILE      VALUE "10".

       77 STARTING-HOUR   PIC 99.
         88 AFTERNOON-HOURS  VALUES ARE 12 THRU 18.

       77 DECISION-FLAG   PIC X.
         88  YES-DECISION  VALUE "Y"
         WHEN SET TO FALSE "N".

       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
           MOVE 13 TO STARTING-HOUR.
           MOVE 10 TO FILE-STATUS.
           MOVE "X" TO DECISION-FLAG.

           IF END-OF-FILE
             DISPLAY "END-OF-FILE" LINE 9 COL 10
           END-IF.

           IF AFTERNOON-HOURS
             DISPLAY "AFTERNOON!" LINE 10 COL 10
           END-IF.

           SET YES-DECISION TO FALSE.
           DISPLAY "DECISION-FLAG: " LINE 11 COL 10.
           DISPLAY DECISION-FLAG  LINE 11 COL 25.
```

General Format
Format 1

The Format 1 VALUE clause is used to initialize data values when the program is initially loaded into memory.

```
[01-LEVEL TO 49-LEVEL, 77-LEVEL DATA-ITEM ]
        [ { VALUE IS }   {literal-1                    } ] .
```

```
           DISPLAY "VALUE2 FINISHED!" LINE 15 COL 10.
           ACCEPT DUMMY LINE 15 COL 30.
           STOP RUN.
```

General Rules

1. The VALUE literal assigned to a condition-name must be of the same data type as the parent data item.
2. THRU and THROUGH are synonyms.
3. Multiple iterations of literal-3 may be listed, without the use of the THRU clause.

   Example: 88 EVEN-NUMBERS VALUES ARE 2,4,6,8,10.

   If the parent data item is set to any of the values in this list, the IF EVEN-NUMBERS condition test would test true.

4. When the literal-3 THRU literal-4 clause is used, the following rules are used to determine what values are included in the THRU statement:

   Literal-3 is a numeric or alphanumeric literal, and sets the low end of the range.
   Literal-4 is a numeric or alphanumeric literal, and sets the high end of the range.

5. If literal-3 is numeric, literal-4 must be numeric. If literal-3 is alphanumeric, literal-4 must be alphanumeric. In each case, literal-4 must have a value greater than literal-3, as determined by the standard collating sequence.

6. The WHEN SET TO FALSE clause causes literal-5 to be moved to the parent data item when the SET statement is used to set the condition-name to FALSE.

Format 3

The Format 3 Value clause is used to set the value of a level-78 constant.

```
[ 78-LEVEL DATA-ITEM]
        [ { VALUE IS }   {literal-6                    } ] .
```

Syntax

1. literal-6 is a numeric or alphanumeric literal which sets the value of the value of the constant named by the 78-level data item.

General Rules

1. The data item associated with literal-6 is a named constant. The named constant can be used interchangeably with literal-6 in the program.

4.9 SCREEN SECTION.

The Screen Section contains descriptions of Screen Section console screen items

```
[ SCREEN SECTION. ]
[ ss-data-level screen-item-description]…
```

4.9.1 SCREEN DESCRIPTION.

A screen description describes all of the elements of a screen, and the attributes of each element.

General Format

```
SCREEN SECTION.
level-number [ screen-name ]
[ LINE line-number [ [ IS ] [ {PLUS } ] numeric-identifier ] ]
                               {  +  }
                               {MINUS}
                               {  -  }

[ {COLUMN} col-number [ [ IS ] [ {PLUS } ] numeric-identifier ] ]
    {COL   }                     {  +  }
                                 {MINUS}
                                 {  -  }

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

  [ FROM from-identifier ]
  [ TO to-identifier ]
  [ USING using-identifier ]

  [ [ USAGE IS ] DISPLAY ]

  [ [SIGN IS] {LEADING } SEPARATE CHARACTER ]
              {TRAILING}

  [ OCCURS integer TIMES ]

  [ {JUSTIFIED} RIGHT ]
    {JUST }

  [ BLANK WHEN ZERO ]

  [ SIZE IS length-in-bytes ]

  [ VALUE IS value-lit ]

  [ BLANK {LINE  }   ]
```

```
          {SCREEN}

  [ ERASE {EOL} ]
          {EOS}

  [ {FOREGROUND_COLOR } IS fg-color-val
    {FOREGROUND-COLOUR}

  [ {BACKGROUND_COLOR } IS bg-color-val
    {BACKGROUND-COLOUR}

  [ UNDERLINE ]

  [ OVERLINE ]

  [ {HIGHLIGHT} ]
    {HIGH     }

  [ {LOWLIGHT} ]
    {LOW     }

  [ BLINK ]

  [ REVERSE-VIDEO ]

  [ {AUTO           } ]
    {AUTO-SKIP     }
    {AUTO-TERMINATE}

  [[NO] {BELL} ]
        {BEEP}

  [ {SECURE } ]
    {NO-ECHO}

  [ {REQUIRED   } ]
    {EMPTY-CHECK}

  [ {FULL        } ]
    {LENGTH-CHECK}

  [ PROMPT  [CHARACTER IS prompt-literal ]  ]

  [ CONTROL in [control-var] ].
```

Syntax

A screen-description consists of a series of screen description entries. Screen Description entries are described below:

General Rules

The General Rules for the Screen Description clauses are described below.

4.9.2 SCREEN DESCRIPTION ENTRIES.

PICTURE Clause

General Format

The PICTURE Clause is described in 4.8.9 PICTURE Clause.

USING Clause

General Format

```
[ USING data-1 ]
```

Syntax

1. **data-n** is a data item.

General Rules

1. The USING data-1 clause implies a MOVE from the target data -1 to the subject of the USING clause when a DISPLAY [Screen] statement is executed, and a MOVE TO the target data-1 when an ACCEPT [Screen] statement is executed.
2. The target data-1 must have the same PICTURE clause as the subject item declared in the Screen Section.
3. The target data-1 must be defined in the file section, working-storage section, local-storage section or linkage section.

FROM Clause

General Format

```
[ FROM identifier-1 ]
```

Syntax

1. identifier-n is a data element, literal, or data returned from a function call.

General Rules

1. The FROM identifier-1 clause implies a MOVE from the target identifier -1 to the subject of the TO clause when a DISPLAY [Screen] statement is executed.

2. The target identifier-1 must have the same PICTURE clause as the subject of the FROM clause.
3. The target identifier-1 must be defined in the file section, working-storage section, local-storage section or linkage section.

TO Clause

General Format

```
[ TO data-1 ]
```

Syntax

1. data-1 is a data item.

General Rules

1. The TO data-1 clause implies a MOVE from the subject of the TO clause to the data-1 item that is the target of the TO clause when an ACCEPT [Screen] statement is executed.
2. The target data-1 must have the same PICTURE clause as the subject of the TO clause.
3. The target data-1 must be defined in the file section, working-storage section, local-storage section or linkage section.

USAGE Clause

General Format

```
[ [USAGE IS] DISPLAY ]
```

Syntax

The USAGE IS DISPLAY clause is described in 4.8.18 Usage Clause.

General Rules

1. When the USAGE IS DISPLAY clause is entered on a group item, it applies to elements that are subordinate to the group.

SIGN Clause

General Format

```
[ [ SIGN IS ] {LEADING } [ SEPARATE CHARACTER ] ]
              {TRAILING}
```

General Rules

The SIGN Clause is described in 4.8.15 Sign Clause.

SCREEN OCCURS Clause

The SCREEN OCCURS clause causes the DISPLAY, or ACCEPT of fields in a table to be repeated a number of times specified by a numer literal or data item. The SCREEN OCCURS clause can be used to format a table on a screen, using LINE and COL attributes with relative positioning.

General Format

```
[ OCCURS numeric-1 TIMES ]
```

Syntax

1. numeric-1 is a literal or data item that is numeric.
2. The OCCURS clause may not be used at the 01-level of the Screen description.
3. The OCCURS clause may be used to create a two-dimensional table, as shown in the example below.
4. When referring to a data item, for purposes of input, or output, the data item must be declared with an OCCURS phrase, and the number numeric-1 must be the same in the data table, and in the Screen Section OCCURS.

General Rules

1. For a summary of the General Rules that apply to the OCCURS clause, see Paragraph 5.8.8 Occurs Clause.
2. Data items declared under an OCCURS clause must not contain subscripts.
3. The effect of the DISPLAY of a table within a SCREEN SECTION OCCURS, is to cause the items in the table to be transferred to the screen, assuming their relative positions within the table.
4. The effect of the ACCEPT of a table within a SCREEN SECTION OCCURS is to cause the items on the Screen to be transferred to the table, assuming their relative positions within the table.

Examples of usage of 1-dimensional and 2-dimensional tables, and of the display of a literal with a OCCURS phrase are included in the Code Sample below.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SCR-OCCURS.
       ENVIRONMENT DIVISION.
      *
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  COLOR-TBL.
         03  COLOR-ITEM OCCURS 5 TIMES     PIC X(6).
      *
       01  VEGGIE-TBL.
         03  OCCURS 5 TIMES.
           05  VEGGIE-NAME    PIC X(10).
           05  VEGGIE-QTY     PIC 9(5).

       77 WS-DUMMY PIC X.
       SCREEN SECTION.
       01  SCREEN-1.
         03  "COLORS:" PIC X(7) LINE 2 COL 2.
         03  LINE 2 COL 10.
         03  OCCURS 5 TIMES USING COLOR-ITEM PIC X(6), COLUMN + 2.
      *
       01  SCREEN-2.
         03  "VEGETABLES      QUANTITY" PIC X(24) LINE 5 COL 2.
         03  LINE 6 COL 2.
         03  OCCURS 5 TIMES.
           05  USING VEGGIE-NAME PIC X(10) LINE + 1 COL 2.
           05  USING VEGGIE-QTY  PIC 9(5) COLUMN + 8
                     BLANK WHEN ZERO
                     PROMPT CHARACTER IS " ".
      *
       01  SCREEN-3.
         03  LINE 14 COL 2.
         03  OCCURS 10 TIMES.
           05 "*" PIC X.
      *
       PROCEDURE DIVISION.
       MAIN-LOGIC.
           DISPLAY SCREEN-1.
           ACCEPT SCREEN-1.

           DiSPLAY SCREEN-2.
           ACCEPT SCREEN-2.

           DISPLAY SCREEN-3.
           ACCEPT WS-DUMMY LINE 14 COL 25.
           STOP RUN.
```

JUSTIFIED Clause

The JUSTIFIED RIGHT clause causes a MOVE of data to an ALPHABETIC or ALPHANUMERIC data item to align at the right-most character position of the data element.

General Format

```
[ {JUSTIFIED} RIGHT ]
  {JUST}
```

General Rules

The JUSTIFIED Clause is described in 4.8.6 JUSTIFIED Clause.

VALUE Clause

The Screen Section VALUE clause is used to initialize data values when the program is initially loaded into memory.

General Format

```
[ VALUE IS value-lit ]
```

Syntax

1. value-lit is a numeric or alphanumeric literal or data item.

General Rules

The VALUE Clause is described in 4.8.19 VALUE Clause.

4.9.3 SCREEN ATTRIBUTES.

CONTROL Clause

The CONTROL Clause is recognized by the compiler, but is not fully implemented.

General Format

```
[ CONTROL IS control-var ]
```

Syntax

control-var is an alphanueric data item.

AUTO

General Format

```
[ {AUTO           } ]
  {AUTO-SKIP      }
  {AUTO-TERMINATE}
```

General Rules

1. AUTO, AUTO-SKIP, and AUTO-TERMINATE are synonyms.
2. If the AUTO attribute is described on a group level, it applies to all of the screen items subordinate to that group.
3. The AUTO attribute moves the cursor automatically to the next input field when the current input field is full.
4. If the AUTO attribute is described on the last input field on the screen, and there is no next input field, then the effect of the AUTO attribute is to cause the ACCEPT to terminate.

BACKGROUND-COLOR

General Format

```
[ {BACKGROUND_COLOR } IS numeric-1
  {BACKGROUND-COLOUR}
```

Syntax

1. numeric-n is a literal or data item that is numeric.

General Rules

1. BACKGROUND-COLOR and BACKGROUND-COLOUR are synonyms.
2. If the BACKGROUND-COLOR attribute is described on a group level, it applies to all of the screen items subordinate to that group.
3. BACKGROUND-COLOR is expressed in an integer value between 0 and 7, with numbers corresponding to colors as described in the color-table below:

Integer	Color
0	Black
1	Blue
2	Green

3	Cyan
4	Red
5	Magenta
6	Yellow
7	White

BELL

General Format

```
[ [NO] {BELL} ]
       {BEEP}
```

General Rules

1. BELL and BEEP are synonyms.
2. Some operating systems do not support the BELL clause. On operating systems that do not support the BELL clause, the BELL clause is treated as commentary.
3. The BELL clause causes the screen field described with the BELL clause to be “beep” when the Screen item is referenced.
4. The NO BELL / NO BEEP phrase suppresses the warning “beep” when the input field is entered.

BLANK

General Format

```
[ BLANK {LINE  }   ]
        {SCREEN}
```

General Rules

1. The BLANK LINE clause causes the entire line on which the screen item is described to be cleared before executing the DISPLAY of that line.
2. The BLANK SCREEN clause causes the entire screen to be cleared before executing the DISPLAY of the screen.
3. The BLANK LINE and BLANK SCREEN clauses are ignored during an ACCEPT statement.

BLINK

General Format

```
[ BLINK ]
```

General Rules

1. Some operating systems do not support the BLINK clause. On operating systems that do not support the BLINK clause, the BLINK clause is treated as commentary.
2. The BLINK clause causes the screen field described with the BLINK clause to blink when the Screen item is referenced.

COLUMN

General Format

```
[ {COLUMN} numeric-1 [ [ IS ]  [ {PLUS } ] numeric-2 ] ]
  {COL   }                       {  +  }
                                 {MINUS}
                                 {  -  }
```

Syntax

1. numeric-1 is a numeric literal or data item.
2. numeric-2 is a numeric literal or data item.

General Rules

1. The COLUMN clause establishes the horizontal position at which to start a DISPLAY or ACCEPT statement, while the LINE clause establishes the vertical position for the given Screen display.
2. The PLUS numeric-2 and MINUS numeric-2 clauses cause position to be established relative to the position of the preceding screen item. PLUS numeric-2 causes the relative column position to increase by the value of integer-2. MINUS numeric-2 causes the relative column position to decrease by the value of integer-2.

ERASE

General Format

```
[ ERASE {EOL} ]
        {EOS}
```

General Rules

1. EOL and END OF LINE are synonyms.

2. EOS and END OF SCREEN are synonyms.
3. The ERASE clause causes a portion of the screen to be erased before the DISPLAY statement transfers data to the screen. The portion of the screen to be erased begins at the position of the screen item declaring the ERASE clause, and proceeds as follows:
4. ERASE EOL erases the screen from the position of the current field to the end of the current line.
5. ERASE EOS erases the screen from the position of the current field to the end of the screen.

FOREGROUND-COLOR

General Format

```
[ {FOREGROUND_COLOR } IS numeric-1
  {FOREGROUND-COLOUR}
```

Syntax

1. numeric-n is a literal or data item that is numeric.

General Rules

1. FOREGROUND-COLOR and FOREGROUND-COLOUR are synonyms.
2. If the FOREGROUND-COLOR attribute is described on a group level, it applies to all of the screen items subordinate to that group.
3. FOREGROUND-COLOR is expressed in an integer value between 0 and 7, with numbers corresponding to colors as described in the color-table below:

Integer	LOWLIGHT COLOR	HIGHLIGHT COLOR
0	Black	Dark Grey
1	Dark Blue	Bright Blue
2	Dark Green	Bright Green
3	Dark Cyan	Bright Cyan
4	Dark Red	Bright Red
5	Dark Magenta	Bright Magenta
6	Brown	Yellow
7	Light Grey	White

FULL

General Format

```
[ {FULL         } ]
  {LENGTH-CHECK}
```

General Rules

The FULL clause is is recognized, and syntax is validated. However, the FULL clause is otherwise treated as commentary.

HIGHLIGHT

General Format

```
[ {HIGHLIGHT} ]
  {HIGH     }
```

General Rules

1. HIGHLIGHT and HIGH are synonyms.
2. Some operating systems do not support the HIGHLIGHT clause. On operating systems that do not support the HIGHLIGHT clause, the HIGHLIGHT clause is treated as commentary.
3. The HIGHLIGHT clause causes the screen field described with the HIGHLIGHT clause to be displayed with high intensity when the Screen item is referenced.

LINE

General Format

```
[ LINE numeric-1 [ [ IS ]   [ {PLUS } ] numeric-2 ] ]
                              {  +  }
                              {MINUS}
                              {  -  }
```

Syntax

1. numeric-n is a literal or data item that is numeric.

General Rules

1. The LINE clause establishes the vertical position at which to start a DISPLAY or ACCEPT statement, while the COLUMN clause establishes the horizontal position on the given LINE

position.

2. If there is no LINE clause, then LINE 1 is assumed, with line 1 being the first line of the Screen described. Note that the Screen described may not be positioned at line 1 of the terminal.
3. The PLUS numeric-2 and MINUS numeric-2 clauses cause position to be established relative to the position of the preceding screen item. PLUS numeric-2 causes the relative line position to increase by the value of integer-2. MINUS numeric-2 causes the relative line position to decrease by the value of integer-2.

LOWER-CASE

The LOWER-CASE attribute changes data entry in Screen Section input fields to lower-case.

General Format

```
LOWER-CASE
```

General Rules

There are no General Rules.

LOWLIGHT

General Format

```
[ {LOWLIGHT} ]
  {LOW     }
```

General Rules

1. LOWLIGHT and LOW are synonyms.
2. Some operating systems do not support the LOWLIGHT clause. On operating systems that do not support the LOWLIGHT clause, the LOWLIGHT clause is treated as commentary.
3. The LOWLIGHT clause causes the screen field described with the LOWLIGHT clause to be displayed with low intensity when the Screen item is referenced.

OVERLINE

General Format

```
[ OVERLINE ]
```

General Rules

1. Some operating systems do not support the OVERLINE clause. On operating systems that do not support the OVERLINE clause, the OVERLINE clause is treated as commentary.
2. The OVERLINE clause causes characters in a field described with the OVERLINE clause to be OVERLINE'd when the Screen item is referenced.

PROMPT

General Format

```
[ PROMPT  [CHARACTER IS prompt-literal ]  ]
```

Syntax

1. prompt-literal is a one-character alphanumeric literal, which is used as the "prompt" character.

General Rules

1. The PROMPT CHARACTER is displayed for all input fields when a SCREEN is DISPLAY'ed.
2. The default PROMPT CHARACTER is the underscore.
3. The PROMPT CHARACTER clause provides the ability to substitute another character for the underscore as the default PROMPT character in an input field.
4. The PROMPT phrase causes the input field to be filled with the specified prompt character prior to entry. If the prompt character is omitted, underscores are used.
5. The PROMPT CHARACTER is a display attribute, and is not transferred back to the TO/USING field associated with the input field after the ACCEPT has terminated.

REQUIRED

General Format

```
[ {REQUIRED   } ]
  {EMPTY-CHECK}
```

General Rules

1. The REQUIRED clause indicates that an input field must have non-space character data entered before the user may proceed to the next input field, or terminate the ACCEPT.
2. REQUIRED and EMPTY-CHECK are synonyms.

REVERSE-VIDEO

General Format

```
[ REVERSE-VIDEO ]
```

General Rules

1. Some operating systems do not support the REVERSE-VIDEO clause. On operating systems that do not support the REVERSE-VIDEO clause, the REVERSE-VIDEO clause is treated as commentary.
2. The REVERSE-VIDEO clause causes the foreground and background colors described, or implicit for the characters in a field described with the REVERSE-VIDEO clause to be reversed when the Screen item is referenced.

SECURE

General Format

```
[ {SECURE } ]
  {NO-ECHO}
```

General Rules

1. The SECURE attribute inhibits the display of data that is entered into a data entry fields.
2. When an ACCEPT is performed on a field with the SECURE attribute, the field is DISPLAY'ed as a string of asterisks. The process of data entry moves the cursor, but does not otherwise alter the DISPLAY'ed view of the field.
3. The data entered into a SECURE field is stored in ithe field into which it was entered, and may be viewed in a debugger. It is not encrypted in any way.

SIZE

General Format

```
[ SIZE  ] [ IS ] length-in-bytes
```

General Rules

1. The SIZE attribute describes the number of bytes that are returned to the screen.
2. If SIZE is less than the length of the data item, then the space is padded with SPACES.

UNDERLINE

General Format

```
[ UNDERLINE ]
```

General Rules

1. Some operating systems do not support the UNDERLINE clause. On operating systems that do not support the UNDERLINE clause, the UNDERLINE clause is treated as commentary.
2. The UNDERLINE clause causes characters in a field described with the UNDERLINE clause to be UNDERLINE'd when the Screen item is referenced.

UPPER-CASE

The UPPER-CASE attribute changes data entry in Screen Section input fields to upper-case.

General Format

```
UPPER-CASE
```

General Rules

There are no General Rules.

5 PROCEDURE DIVISION.

The Procedure Division contains the statements that the program executes, and describes data items that may be passed to the program from a CALLing program, or from the command-line.

5.1 PROCEDURE DIVISION Clause

General Format

```
PROCEDURE DIVISION
```

```
      [ { USING    }  ]
        { CHAINING }

      [ { BY REFERENCE } ]
        { BY VALUE     }
        { BY CONTENT   }

      [ [ UNSIGNED ] SIZE [ IS ] { AUTO    } ]
                                 { DEFAULT }
                                 { integer }

      [ [ OPTIONAL ] { param } ... ]

      [ RETURNING ] [ { param } ... ].


   [ DECLARATIVES.
     { procedure-sect SECTION [segment-no] .
              use-statement
     [ procedure-para.
              [ procedure-statement ] ... ] ... } ...
   END DECLARATIVES. ]

    { procedure-sect SECTION [segment-no] .
    [ procedure-para.
        [ procedure-statement ] ... ] ... } ...
```

Syntax

The clauses supported in the Procedure Division statement are described below.

5.1.1 USING/CHAINING Clause

USING is followed by a parameter or list of parameters, whose order matches the order of the parameters listed in a CALL statement in a CALLing program.

General Format

```
   [ { USING    }  ]
     { CHAINING }
```

General Rules

1. USING and CHAINING clauses are used to describe data elements that are being passed to a program that is serving as a subroutine.
2. Data elements described by the USING clause correspond to data elements named in a CALL statement invoking the subroutine.
3. Data elements described by the CHAINING clause correspond to data elements named in a CHAIN statement invoking the subroutine.
4. All data items expressed in USING / CHAINING clauses must be defined in the LINKAGE

SECTION of the program.

5. Note that the CHAIN statement and CHAINING clause are recognized, and syntax is validated. However, the CHAIN statement /CHAINING clause are otherwise treated as commentary.
6. The usage of a redefined field int the Linkage Section is allowed. The redefined field may subsequently be referenced in the PROCEDURE DIVISION USING clause.

5.1.2 BY Clause

The BY clause describes the manner in which data is passed to a subprogram.

General Format

```
{ BY REFERENCE }
{ BY VALUE     }
{ BY CONTENT   }
```

General Rules

1. The BY REFERENCE clause indicates that the address of the data item has been passed through the Linkage Section. Updates to the data item are made directly to the memory address of the data item, and changes to the value of the data item will be seen in the CALL'ing program.
2. The BY VALUE clause indicates that the VALUE, rather than memory address of the item is passed through the Linkage Section. Updates to the data item will only be seen locally in the sub-program, and will not be returned through the data item to the CALL'ing program.
3. The BY CONTENT clause indicates that a copy of the address of the data item has been passed through the Linkage Section. Updates to the data item will only be seen locally in the sub-program, and will not be returned through the data item to the CALL'ing program.
4. BY REFERENCE is assumed as the default if no BY clause is present.

5.1.3 SIZE Clause

The SIZE clause describes the size, in bytes, of a data item.

General Format

```
[ [ UNSIGNED ] SIZE [ IS ] { AUTO    } ]
                           { DEFAULT }
                           { integer }
```

General Rules

1. The SIZE clause is only allowed when passing BY VALUE.
2. SIZE IS AUTO causes the argument size to be matched to the size of the corresponding data item in the CALL'ing program.
3. SIZE IS DEFAULT causes the argument size to be set to 4.
4. SIZE IS [integer] can be used, where valid values for integer are 1, 2, 4, and 8.

5.1.4 OPTIONAL Clause

The OPTIONAL clause may be used to indicate a data item being passed BY REFERENCE is not required.

General Format

```
[ OPTIONAL ]
```

General Rules

There are no General Rules.

5.1.5 RETURNING Clause

The RETURNING clause describes data items to be returned to the CALL'ing program.

General Format

```
[ RETURNING ] [ { param } ... ].
```

Syntax

1. param must be declared in the Linkage Section.

General Rules

1. The RETURNING clause causes the value of *param* to be returned to the data item named in the RETURNING clause of the CALL'ing program.

5.2 DECLARATIVES.

General Format

```
[ DECLARATIVES.
  { section_name   SECTION [opt_segment].
           use-statement
  [ procedure-para.
           [ procedure-statement ] ... ] ... } ...
END DECLARATIVES. ]
```

Syntax

1. section-name is a user-defined word that is the name of a section.
2. opt-segment is treated as commentary.
3. The use-statement is a statement that begins with the USE verb.
4. procedure-para is a paragraph name.
5. procedure-statement is an imperative or conditional statement.

General Rules

1. The DECLARATIVES section is used to assign procedural statements to be executed under different failure scenarios, as described in the USE statement.
2. For details on the usage of the USE statement, see 5.3.46 USE Statement General Rules.

Code Sample

```
...
PROCEDURE DIVISION.
DECLARATIVES.
IN-FILE-ERR SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON INPUT.
REPORT-ERR-PARA.
    DISPLAY"ERROR IN INPUT FILE. ".
    DISPLAY"FILE-STATUS ITEM IS  "FILE-STATUS.
    CLOSE FILE-1.
    ACCEPT DUMMY.
    STOP RUN.
END DECLARATIVES.
...
```

5.3 PROCEDURE DIVISION STATEMENTS.

PROCEDURE DIVISION statements must begin with a reserved word called a verb.

General Format

```
{ procedure-sect SECTION [segment-no] .
[ procedure-para.
    [ procedure-statement ] ... ] ... } ...
```

General Rules

The General Rules for the different Procedure Division statements are described below.

Common General Rules are General Rules that apply to more than one Statement.

5.3.1 Common General Rules

ROUNDED

1. The effect of the ROUNDED clause is to cause the least significant digit to be increased by 1 if the next least significant digit of the intermediate result is greater than or equal to 5.
2. When there is no ROUNDED clause specified, the next least significant digit, and all subsequent digits in the intermediate result are truncated.

ON SIZE ERROR

1. The SIZE ERROR exception is triggered if the ON SIZE ERROR clause is present and if the receiving field is not large enough to accommodate the result of an arithmetic function.
2. In the absence of an ON SIZE ERROR clause, if the receiving field is not large enough to accommodate the result of the ADD function, the high-order digits that cannot be accommodated are truncated. For example, if you ADD 6 TO 6, and attempt to store the result in a PIC 9 field, the result will be 2.
3. The ON SIZE ERROR clause provides the programmer with a way to respond to the SIZE ERROR exception by performing a statement, or a series of statements.
4. The NOT ON SIZE ERROR condition exists if the NOT ON SIZE ERROR clause is present, and a SIZE ERROR exception is not triggered.
5. The NOT ON SIZE ERROR clause provides the programmer with a way to respond to the NOT ON SIZE ERROR condition by performing a statement, or a series of statements.

ARITHMETIC ORDER OF EVALUATION

1. Arithmetic expressions within parentheses are evaluated first.
2. Within nested sets of parentheses, the innermost set of parentheses is evaluated first. Sets of parentheses inside a nested set of parentheses then are evaluated from innermost to outermost.
3. Inside parentheses, or following the complete evaluation of all expressions within parentheses, or in expressions where there are no parentheses, arithmetic expressions evaluate arithmetic operators in the following order:
    a. Unary. The Sign (+ or -) is applied . The minus "-" sign has the effect of causing the expression to be multiplied by -1. The plus "+" sign has the effect of causing the expression to be multiplied by +1.
    b. Exponents. Exponentiation is applied. The exponentiation "**" sign has the effect of raising the expression to the exponent ial value expressed after the exponentiation sign.
    c. Multiplication and Division. Where there are no parentheses to establish order of execution, an arithmetic expression containing successive multiplication and division operators is evaluated from left to right.

d. Addition and Subtraction . Where there are no parentheses to establish order of execution, an arithmetic expression containing successive addition and subtraction operators is evaluated from left to right.

4. The number of left parentheses in an arithmetic expression must equal the number of right parentheses in the arithmetic expression.

ARITHMETIC CALCULATIONS

1. Certain common general rules apply to each of the arithmetic statements ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT.
2. The maximum size of a numeric field is 36 digits.
3. The compiler constructs data items as needed to hold intermediate results in an arithmetic operation.
4. The final operation in an arithmetic expression transfers the final value to the receiving field. If the receiving field is too small to accommodate the result, and if there is no ON SIZE ERROR clause, then truncation occurs on the high-order digits before the decimal point, and on the low-order digits after the deimal point.

Rules for identifying CORRESPONDING data elements

1. A data element **cannot** be considered a CORRESPONDING data element if:
   a. The data element name is not matched.
   b. The data element is described as FILLER.
   c. The data element is described as a REDEFINES of another data item.
   d. The data element is described as a RENAMES of another data item.
   e. The data element has an OCCURS clause.

2. If none of the above apply, then a data element in a target-group is considered a CORRESPONDING data element if:
   a. It has the same name as a data element in the src-group-item, and,
   if it is has one or more parent data items that are not the src-group-item, then all of the parent data items in the src-group-item and target-group-item also have the same names.

5.3.2 ACCEPT Statement

The ACCEPT Statement gives the program access to data either through direct user interaction, or through interaction with the operating environment.

General Format

Format 1 ACCEPT data-field

The Format 1 ACCEPT Statement describes the ACCEPT of a field described in the Data Division.

```
ACCEPT data-1
```

```
[ AT ]    LINE          numeric-1
[ AT ]  { COLUMN   }    numeric-2
        { POSITION }

[ WITH [NO] {BELL}  ]
           {BEEP}

[ WITH BLINK ]

[ { WITH HIGHLIGHT } ]
  { WITH LOWLIGHT  }

[ WITH REVERSE-VIDEO ]

[ { WITH UNDERLINE } ]
  { WITH OVERLINE  }

[ FOREGROUND-COLOR IS numeric-3 ]
[ BACKGROUND-COLOR IS numeric-4 ]

[ SCROLL { UP   }   ]
         { DOWN }

[ AUTO ]

[ FULL ]

[ REQUIRED ]

[ SECURE ]

[ UPDATE ]

[ PROMPT [ CHARACTER IS literal-1 ] ]

[ TIMEOUT timeout-value ]

[ ON EXCEPTION [crtstatus-var] statement-1      ]
[ NOT ON EXCEPTION statement-2  ]
[ END-ACCEPT ]
```

Syntax

1. data-n is a data item.
2. numeric-n is a literal or data item that is numeric.
3. literal-n is a character string.
4. statement-n is an imperative statement.
5. crtstatus-var is a numeric data field, declared either as PIC 9(4) or PIC 9(6).
6. timeout-value is a literal or data item that is numeric.

General Rules

Details on all of the SCREEN attributes below are described in 4.9.2 Screen Description Entries.

1. The LINE clause positions the cursor vertically on the screen.
2. The COLUMN clause positions the cursor horizontally on the screen.

3. The BELL attribute generates an audible “beep” sound when the input field is entered.
4. The BLINK attribute causes the screen item to blink.
5. The HIGHLIGHT attribute causes the screen item to display in high intensity.
6. The LOWLIGHT attribute causes the screen item to display in low intensity.
7. The REVERSE-VIDEO attribute reverses the foreground and background colors in the display of the screen item.
8. The UNDERLINE attribute underlines the screen item.
9. The OVERLINE attribute places a line over the screen item.
10. The FOREGROUND-COLOR attribute assigns one of 8 colors to the foreground text.
11. The BACKGROUND-COLOR attribute assigns one of 8 colors to the background color.
12. The SCROLL phrase scrolls the screen up or down be a designated number of lines.
13. The AUTO phrase automatically moves to the next data entry field when the current field is full.
14. The FULL phrase requires that the field be FULL before it is exited.
15. The REQUIRED phrase requires that the field have data entered before it is exited.
16. The SECURE attribute allows data entry that does not echo to the screen.
17. The UPDATE phrase causes the value of the field associated with the screen item to update the screen field prior to data entry.
18. The PROMPT phrase allows the user to provide different characters for the PROMPT of a field.
19. The TIMEOUT phrase allows for an ACCEPT statement to be automatically terminated after a number of seconds, as defined in [timeout-value. If no data has been entered into the ACCEPT statement in the time defined in [timeout-value], then the ACCEPT statement terminates, generating an exception condition, and an exception value.
20. If any data is entered into the ACCEPT before the designated timeout value, then the timeout timer is restarted.
21. If no data is entered into the ACCEPT before the designated timeout value, then an exception condition occurs, and the COB-SCR-TIMEOUT exception value of 9001 is generated into the CRT STATUS variable.

As an example, with a ACCEPT TIMEOUT of 5 seconds:

```
ACCEPT data-1 TIMEOUT 5
  ON EXCEPTION
    IF CRT-STATUS = COB-SCR-TIMEOUT
      DISPLAY “TIMED OUT” LINE 10 COL 10
    END-IF
END-ACCEPT.
```

The ON EXCEPTION condition is triggered when an exception key is pressed, or an exception condition occurs, terminating the ACCEPT statement. When an ON EXCEPTION condition is triggered, and the ON EXCEPTION phrase is included in the ACCEPT statement:

The statement-list insided the scope of the ON EXCEPTION clause is executed. Control passes to the next statement after the ACCEPT statement.

The NOT ON EXCEPTION condition is triggered when the ACCEPT statement is terminated normally. When an ACCEPT statement is terminated normally, the NOT ON EXCEPTION clause causes the statement –list inside the scope of the NOT ON EXCEPTION clause to be executed. Control then passes to the next statement after the ACCEPT statement.

Compiler Flags

The following compiler flags affect the behaviour of the field level or Screen Section ACCEPT:

-faccept-with-auto	causes the WITH AUTO clause to be assumed on field-level ACCEPT statements.
-faccept-with-update	causes the WITH UPDATE clause to be assumed on field-level ACCEPT statements.

Compiler Configuration File Flags

The following compiler configuration file flags affect the behaviour of the field level or Screen Section ACCEPT:

accept-with-update:yes | no	mimics the behavior of –faccept-with-update.
crtstatus-map: cit-value user-value	Allows the user to re-map default crt status values for function keys, and other keystrokes. If no crtstatus-map is defined , CRT STATUS values are converted to PIC 9(4) and copied into the crt-status-var.
screen-exceptions	mimics the behavior the the environment variable COB_SCREEN_EXCEPTIONS. Enables the use of Page-Up, Page-Down, Up Arrown, Down Arrow keys on a field-level ACCEPT statement.
screen-raw-keys	mimics the behavior of the environment variable COB_SCREEN_RAW_KEYS. Enables the use of the Home, End, Insert, Delete, and Erase EOL keys. Pressing the DEL key deletes the current character and moves the cursor one character to the left. A SPACE is inserted at the end of the field. The EOL key erases to the end of the line. The BACKSPACE key shifts the contents of the field one character to the left, beginning at the current character.

Runtime environment variables

The following runtime environment variables affect the behaviour of the field-level ACCEPT:

COB_SCREEN_EXCEPTIONS when set to Y, enables the use of the Page Up, Page Down, Up Arrow, and Down Arrow keys on field-level ACCEPT statements.

COB_SCREEN_UPDATE_FIRST_KEY_ERASE when set to Y, causes the first key pressed in an input field to record the keystroke, and erase the rest of the field, for all field-level ACCEPT WITH UPDATE statements.

COB_SCREEN_DISABLE_REFORMAT when set to Y, disables the COB_SCREEN_UPDATE_FIRST_KEY_ERASE behaviour.

Format 2 ACCEPT screen

The Format 2 ACCEPT Statement describes the ACCEPT of a screen described in the Screen Section.

General Format

```
ACCEPT screen-1
       [ AT ] LINE       numeric-1
       [ AT ] { COLUMN   }      numeric-2
              { POSITION }

       [ WITH BELL ]

       [ WITH BLINK ]

       [ { WITH HIGHLIGHT } ]
         { WITH LOWLIGHT  }

       [ WITH REVERSE-VIDEO ]

       [ { WITH UNDERLINE } ]
         { WITH OVERLINE  }

       [ FOREGROUND-COLOR IS numeric-3 ]
       [ BACKGROUND-COLOR IS numeric-4 ]

       [ SCROLL { UP   }   ]
                { DOWN }

       [ AUTO ]
```

```
[ FULL ]

[ REQUIRED ]

[ SECURE ]

[ UPDATE ]

[ PROMPT [ CHARACTER IS literal-1 ] ]


[ TIMEOUT timeout-value ]

[ ON EXCEPTION [crtstatus-var] statement-1      ]
[ NOT ON EXCEPTION statement-2  ]
[ END-ACCEPT ]
```

Syntax

1. screen-1 is a screen declared in the Screen Section .
2. numeric-n is a literal or data item that is numeric.
3. literal-n is a character string.
4. statement-n is an imperative statement.
5. crtstatus-var is a numeric data field, declared either as PIC 9(4) or PIC 9(6).
6. timeout-value is a literal or data item that is numeric.

General Rules

Details on all of the SCREEN attributes below are described in 4.9.2 Screen Description Entries.

1. The LINE clause positions the cursor vertically on the screen.
2. The COLUMN clause positions the cursor horizontally on the screen.
3. The BELL attribute generates an audible “beep” sound when the input field is entered.
4. The BLINK attribute causes the screen item to blink.
5. The HIGHLIGHT attribute causes the screen item to display in high intensity.
6. The LOWLIGHT attribute causes the screen item to display in low intensity.
7. The REVERSE-VIDEO attribute reverses the foreground and background colors in the display of the screen item.
8. The UNDERLINE attribute underlines the screen item.
9. The OVERLINE attribute places a line over the screen item.
10. The FOREGROUND-COLOR attribute assigns one of 8 colors to the foreground text.
11. The BACKGROUND-COLOR attribute assigns one of 8 colors to the background color.
12. The SCROLL phrase scrolls the screen up or down be a designated number of lines.
13. The AUTO phrase automatically moves to the next data entry field when the current field is full.
14. The FULL phrase requires that the field be FULL before it is exited.
15. The REQUIRED phrase requires that the field have data entered before it is exited.
16. The SECURE attribute allows data entry that does not echo to the screen.
17. The UPDATE phrase causes the value of the field associated with the screen item to update

the screen field prior to data entry.

18. The PROMPT phrase allows the user to provide different characters for the PROMPT of a field.
19. The TIMEOUT phrase allows for an ACCEPT statement to be automatically terminated after a number of seconds, as defined in [timeout-value. If no data has been entered into the ACCEPT statement in the time defined in [timeout-value], then the ACCEPT statement terminates, generating an exception condition, and an exception value.
    a. If any data is entered into the ACCEPT before the designated timeout value, then the timeout timer is restarted.
    b. If no data is entered into the ACCEPT before the designated timeout value, then an exception condition occurs, and the COB-SCR-TIMEOUT exception value of 9001 is generated into the CRT STATUS variable. Note that COB-SCR-TIMEOUT, and out default screen exception values are described in the file screenio.cpy, located in the $COBOLITDIR\copy directory.

    As an example, with a ACCEPT TIMEOUT of 5 seconds:

```
ACCEPT screen-1 TIMEOUT 5
  ON EXCEPTION
    IF CRT-STATUS = 9001
      DISPLAY "TIMED OUT" LINE 10 COL 10
    END-IF
END-ACCEPT.
```

    c. The ON EXCEPTION condition is triggered when an exception key is pressed, terminating the ACCEPT statement. When an ON EXCEPTION condition is triggered, and the ON EXCEPTION phrase is included in the ACCEPT statement:
    d. The statement-list insided the scope of the ON EXCEPTION clause is executed. Control passes to the next statement after the ACCEPT statement.
    e. The NOT ON EXCEPTION condition is triggered when the ACCEPT statement is terminated normally. When an ACCEPT statement is terminated normally, the NOT ON EXCEPTION clause causes the statement –list inside the scope of the NOT ON EXCEPTION clause to be executed. Control then passes to the next statement after the ACCEPT statement.

20. The following runtime environment variables affect the behaviour of the Screen Section ACCEPT:
    a. COB_SCREEN_DISABLE_REFORMAT, when set to Y, disables the reformatting associated by default with the COB_SCREEN_UPDATE_FIRST_KEY_ERASE behavior.
    b. COB_SCREEN_ESC, when set to Y, enables use of the escape key.Note that Page Up, Page Down, Up Arrow, and Down Arrow are enabled by default when ACCEPTing a Screen.
    c. COB_SCREEN_INPUT_BOLDED, when set to Y, causes all input fields to be displayed in bold.

d. COB_SCREEN_INPUT_FILLER, when set to [char], changes the PROMPT character to [char].
e. COB_SCREEN_INPUT_INSERT_TOGGLE, when set to Y, causes the INS key to toggle between Overwrite and Insert mode. By default, pressing the INS key inserts a SPACE at the current cursor position.
f. COB_SCREEN_INPUT_REVERSED, when set to Y, causes all input fields to be displayed in REVERSE.
g. COB_SCREEN_INPUT_UNDERLINED, when set to Y, causes all input fields to be displayed with UNDERLINE.
h. COB_SCREEN_RAW_KEYS, when set to Y, enables use of the HOME, END, Ins, Del, and Erase EOL keys.

Code sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  ACCEPT-1.
      * ACCEPT SCREEN
      * ... LINE {PLUS/+} [ LINE NUMBER ]
      * ... COL {PLUS/+}  [ COL NUMBER  ]
      * PIC X
      * TO  [ WS-FLD-NAME ]
      * USING [ WS-FLD-NAME ]
      * VALUE [ LITERAL-1 ]
      * BLANK SCREEN
      * PROMPT CHARACTER
       AUTHOR.  CAVANAGH.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  WS-CHOICE PIC X VALUE SPACES.
         88  VALID-SELECTION     VALUE"B", "S", "F", "H", "T".
       77  WS-ALLSTAR PIC X(40)  VALUE SPACES.

       SCREEN SECTION.
       01  MENU-SCREEN.
         05 VALUE "SELECT A PROFESSIONAL SPORT "
              BLANK SCREEN LINE 02 COL 26.
         05 VALUE "B - BASEBALL" LINE + 2 COL 05.
         05 VALUE "S - SOCCER"   LINE PLUS 2 COL 05.
         05 VALUE "F - FOOTBALL" LINE 08 COL 05.
         05 VALUE "H - HOCKEY"   LINE 10 COL 05.
         05 VALUE "T - TERMINATE PROGRAM" LINE 12 COL 05.
         05 VALUE "ENTER CHOICE:" LINE17 COL PLUS 5.
         05  MENU-ANS-SCR LINE 17 COL + 15
                PIC X TO WS-CHOICE
                PROMPT CHARACTER IS "+".
         05  MENU-ALLSTAR     PIC X(40)  LINE 20 COL 20
                USING  WS-ALLSTAR.
      *
       PROCEDURE DIVISION.
       MAINLINE.
           DISPLAY MENU-SCREEN.
           PERFORM ACCEPT-LOOP UNTIL WS-CHOICE = "T".
           STOP RUN.
      *
       ACCEPT-LOOP.
           ACCEPT MENU-SCREEN.
```

```
           EVALUATE WS-CHOICE
             WHEN "B" MOVE "WILLY MAYS" TO WS-ALLSTAR
             WHEN "S" MOVE "RONALDO" TO WS-ALLSTAR
             WHEN "F" MOVE "PEYTON MANNING" TO WS-ALLSTAR
             WHEN "H" MOVE "BOBBY ORR" TO WS-ALLSTAR
             WHEN "T" CONTINUE
             WHEN OTHER MOVE "INVALID ENTRY" TO WS-ALLSTAR
           END-EVALUATE.
           DISPLAY MENU-ALLSTAR.
```

Format 2 Return Terminal Size characteristics

```
ACCEPT numeric-1 FROM  { LINES   }
                       { COLUMNS }
```

Syntax

1. numeric-n is a 3-digit numeric data item that returns the number of LINES/COLUMNS on the terminal.

General Rules

1. The ACCEPT numeric-1 FROM LINES statement retrieves the number of lines on the terminal console in which the program is executing.
2. The ACCEPT numeric-1 FROM COLUMNS statement retrieves the number of columns on the terminal console in which the program is executing.
3. Values returned represent the height and width characteristics, in characters, of the terminal console in which the program is executing.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  ACCEPT-2.
      * ACCEPT NUMERIC FROM LINES
      * ACCEPT NUMERIC FROM COLUMNS
       AUTHOR.  CAVANAGH.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 TEST-VAR1    PIC 999.
       77 TEST-VAR2    PIC 999.
       77 DUMMY       PIC X.
       SCREEN SECTION.
       PROCEDURE DIVISION.
       MAIN.
           DISPLAY "TESTING ACCEPT VAR FROM LINES" LINE 10 COL 10.
           ACCEPT TEST-VAR1 FROM LINES.
           DISPLAY TEST-VAR1 LINE 12 COL 10.

           DISPLAY "TESTING ACCEPT VAR FROM COLUMNS" LINE 15 COL 10.
           ACCEPT TEST-VAR2 FROM COLUMNS.
           DISPLAY TEST-VAR2 LINE 17 COL 10.
```

```
           DISPLAY "ACCEPT FROM LINES/COLUMNS FINISHED!"
                             LINE 20 COL 10.
           ACCEPT DUMMY LINE 20 COL 45.
           STOP RUN.
```

Format 3 - Return Date-Time, and Command-line arguments

```
ACCEPT data-3 FROM          { DATE } [ YYYYMMDD ]
                            { DAY  }
                            { DAY-OF-WEEK }
                            { TIME }
                            { COMMAND-LINE }
                            { ARGUMENT-NUMBER }
```

Syntax

1. data-n is a data item.

General Rules

1. ACCEPT data-3 FROM DATE returns the date in the format yymmdd.
2. ACCEPT data-3 FROM DATE YYYYMMDD returns the date in the format yyyymmdd.
3. ACCEPT data-3 FROM DAY returns the date in the Julian format YYDDD where DDD represents the ordinal position of the current day of the year.
4. ACCEPT data-3 FROM DAY YYYYMMDD returns the date in the Julian format YYYYDDD where DDD represents the ordinal position of the current day of the year.
5. ACCEPT data-3 FROM DAY-OF-WEEK returns a numeric value between 1 and 7, with 1 representing Monday, 2 representing Tuesday, etc... and 7 representing Sunday.
6. ACCEPT data-3 FROM TIME returns the time in the format HHMMSShh, where:
    a. HH refers to Hours, and is returned as an integer between 0 and 23.
    b. MM refers to Minutes, and is returned as an integer between 0 and 59.
    c. SS refers to Seconds, and is returned as an integer between 0 and 59.
    d. hh refers to hundredths of seconds, and is returned as an integer between 0 and 99.
7. ACCEPT data-3 FROM COMMAND-LINE returns the arguments that were included on the command line after the program name. As an example, the command line below for running the program myprog contains 2 arguments, which are hello world.:

    >cobcrun myprog hello world
    In this example, ACCEPT data-3 FROM COMMAND-LINE returns the string “hello world” into data-3.
8. ACCEPT data-3 FROM ARGUMENT-NUMBER returns the number of arguments on the command line. In the example above, there are 2 arguments, so the integer “2” would be returned.
    a. To determine the ARGUMENT-NUMBER, COBOL-IT parses the command line, treating spaces as delimiters, except when they are enclosed within quotation (“ “)

characters. Words contained within quotation characters are treated as a single argument.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  ACCEPT-3.
      *FORMAT 3 ACCEPT
      * ACCEPT DATA-3 FROM      { DATE } [ YYYYMMDD ]
      *                        { DAY  }
      *                        { DAY-OF-WEEK }
      *                        { TIME }
      *                        { COMMAND-LINE }
      *                        { ARGUMENT-NUMBER }
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 VAR1A         PIC 9(6).
       77 VAR1B         PIC 9(8).
       77 VAR2          PIC 9(5).
       77 VAR3          PIC 9.
       77 VAR4          PIC 9(8).
       77 VAR5          PIC X(10).
       77 VAR6          PIC 9.

       77 DUMMY        PIC X.
       SCREEN SECTION.
       PROCEDURE DIVISION.
       MAIN.
           ACCEPT VAR1A FROM DATE.
           DISPLAY VAR1A LINE 5 COL 10.

           ACCEPT VAR1B FROM DATE YYYYMMDD.
           DISPLAY VAR1B LINE 6 COL 10.

           ACCEPT VAR2 FROM DAY.
           DISPLAY VAR2 LINE 7 COL 10.

           ACCEPT VAR3 FROM DAY-OF-WEEK.
           DISPLAY VAR3 LINE 9 COL 10.

           ACCEPT VAR4 FROM TIME.
           DISPLAY VAR4 LINE 10 COL 10.

           ACCEPT VAR5 FROM COMMAND-LINE.
           DISPLAY VAR5 LINE 11 COL 10.

           ACCEPT VAR6 FROM ARGUMENT-NUMBER.
           DISPLAY VAR6 LINE 12 COL10.

           DISPLAY "ACCEPT-3 FINISHED!" LINE 20 COL10.
           ACCEPT DUMMY LINE 20 COL 45.
           STOP RUN.
      *
```

Format 4 - Return the value of Environment Variables

```
ACCEPT data-4 FROM  ENVIRONMENT "[environment-variable-name]"
```

```
            [ ON EXCEPTION statement-1 ]
            [ NOT ON EXCEPTION statement-2 ]
            [ END-ACCEPT ]
```

Syntax

1. data-n is a data item.
2. environment-variable-name is the name of the environment variable being interrogated.
3. statement-n is an imperative statement.

General Rules

1. ACCEPT data-4 FROM ENVIRONMENT interrogates the current command shell for the value of a given environment-variable-name.
2. If the environment variable named by environment-variable-name does not exist, an EXCEPTION condition is created, and statement-1 is executed in the formulation ON EXCEPTION statement-1
3. EXCEPTION conditions can only be detected through the ON EXCEPTION clause.
4. The ON EXCEPTION clause is not required.
5. The NOT ON EXCEPTION clause is parsed but is otherwise treated as commentary.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  ACCEPT-4.
      * ACCEPT VAR FROM ENVIRONMENT
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77DATA-4    PIC X(30) VALUE SPACES.
       77 DUMMY        PIC X.
       SCREEN SECTION.
       PROCEDURE DIVISION.
       MAIN.
      *FORMAT 3
      *ACCEPT DATA-4 FROM  ENVIRONMENT

           ACCEPT DATA-4 FROM ENVIRONMENT "COBOLITDIR".
           DISPLAY DATA-4 LINE 10 COL 10.

           DISPLAY "ACCEPT-4 FINISHED!" LINE 20 COL 10.
           ACCEPT DUMMY LINE 20 COL 45.
           STOP RUN.
```

Format 5 – Return Argument-value

```
ACCEPT data-5 FROM ARGUMENT-VALUE
    [ ON EXCEPTION statement-1 ]
    [ NOT ON EXCEPTION statement-2 ]
    [ END-ACCEPT ]
```

Syntax

1. data-n is a data item.
2. statement-n is an imperative statement.

General Rules

1. ACCEPT data-5 FROM ARGUMENT-VALUE processes the command line as a series of space-delimited parameters.
2. ACCEPT data-5 FROM ARGUMENT-VALUE can be used iteratively to return the values of all of the arguments on the command line.
3. If there are no arguments on the command line, or if the values of all of the arguments on the command line have been returned, an an EXCEPTION condition is created, and statement-1 is executed in the formulation :
   ON EXCEPTION statement-1
   EXCEPTION conditions can only be detected through the ON EXCEPTION clause.
4. The ON EXCEPTION clause is not required.
5. The NOT ON EXCEPTION clause is parsed but is otherwise treated as commentary.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  ACCEPT5.
      * ACCEPT...FROM ARGUMENT NUMBER/ARGUMENT-VALUE
      * THIS STATEMENT PROCESSES THE COMMAND LINE AS
      * A SERIES OF SPACE-DELIMITED
      * PARAMETERS.  SEE YOUR LANGUAGE REFERENCE FOR DETAILS.
      * RUN COBCRUN ACCEPT-5 ARG1 ARG2 ARG3
      * ACCEPT VAR FROM ENVIRONMENT
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 DATA-1      PIC X(20).
       77 DATA-2      PIC 9.
       77 LINE-NUM    PIC 9.
       77 DATA-3      PIC X(20) VALUE SPACES.
       77 DUMMY       PIC X.
       SCREEN SECTION.
       PROCEDURE DIVISION.
       MAIN.
           ACCEPT DATA-1 FROM COMMAND-LINE.
           ACCEPT DATA-2 FROM ARGUMENT-NUMBER.

           MOVE 2 TO LINE-NUM
           ADD 1 TO DATA-2
```

```
           PERFORM DISPLAY-ARGUMENT-VALUES DATA-2 TIMES

           DISPLAY "ACCEPT5 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.

           STOP RUN.
       *
        DISPLAY-ARGUMENT-VALUES.
           ACCEPT DATA-3 FROM ARGUMENT-VALUE
             ON EXCEPTION
               DISPLAY "THAT IS ALL FOLKS!" LINE LINE-NUM COL 10
           END-ACCEPT.

           IF DATA-3 NOT = SPACES
             DISPLAY DATA-3 LINE LINE-NUM COL 10
             ADD 1 TO LINE-NUM
             INITIALIZE DATA-3
           END-IF.
```

Format 6 – ACCEPT FROM SYSIN/CONSOLE

```
ACCEPT data-6 FROM  { mnemonic-name }
```

Syntax

1. data-n is a data item.
2. mnemonic-name names the hardware device from which data is being transferred by the ACCEPT statement.

General Rules

1. mnemonic-name must be either SYSIN or CONSOLE.

Code Sample

```
        IDENTIFICATION DIVISION.
        PROGRAM-ID.  ACCEPT-6.
       * ACCEPT...FROM SYSIN
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        77 DATA-1       PIC X(20).
        77 DUMMY        PIC X.
        SCREEN SECTION.
        PROCEDURE DIVISION.
        MAIN.
           ACCEPT DATA-1 FROM SYSIN.
           DISPLAY DATA-1 LINE 8 COL 10.

           DISPLAY "ACCEPT-6 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.

           STOP RUN.
```

Format 7 – ACCEPT FROM [WORD]

```
ACCEPT data-7 FROM [WORD]
```

Syntax

1. data-n is a data item.
2. WORD is associated with a hardware device in SPECIAL-NAMES.

General Rules

1. WORD is a user-defined word that is assigned to SYSIN or CONSOLE in SPECIAL-NAMES.

   Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  ACCEPT-7.
       ENVIRONMENTDIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
           SYSIN IS KEYBOARD.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 DATA-1   PIC X(30).
       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
           ACCEPT DATA-1 FROM KEYBOARD.
           DISPLAY DATA-1 LINE 10 COL 10.

           DISPLAY "ACCEPT-7 FINISHED!" LINE 12 COL 10.
           ACCEPT DUMMY LINE 12 COL 30.
           STOP RUN.
```

Format 8- ACCEPT FROM ESCAPE KEY

ACCEPT FROM ESCAPE KEY is used to retrieve the value of the CRT STATUS variable when an ON EXCEPTION condition is triggered on an ACCEPT statement without having to declare CRT STATUS in SPECIAL-NAMES.

General Format

```
ACCEPT data-8 FROM ESCAPE KEY
```

Syntax

1. data-8 is a numeric data item that is 4 bytes in length.

General Rules

1. ACCEPT FROM ESCAPE KEY is used to retrieve the exception value returned on an exception condition.
2. ACCEPT FROM ESCAPE KEY is executed inside the scope of the ON EXCEPTION clause.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  ACCEPT-8.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 CRT-STAT  PIC 9(4).
       77 DATA-1    PIC X(10).
       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
           DISPLAY "DATA 1: " LINE 10 COL 10.
           ACCEPT DATA-1
              ON EXCEPTION
                 ACCEPT CRT-STAT FROM ESCAPE KEY
           END-ACCEPT.

           DISPLAY CRT-STAT LINE 15 COL 10.

           DISPLAY "ACCEPT-8 FINISHED!" LINE 18 COL 10.
           ACCEPT DUMMY LINE 18 COL 30.
           STOP RUN.
```

5.3.3 ADD Statement

The ADD Statement performs an arithmetic add operation on a number of operands and stores the result.

Format 1

```
ADD { numeric-1 } … TO { numeric-2 } …
    [ GIVING numeric-data-1 [ROUNDED] ]
    [ ON SIZE ERROR statement-1 ]
    [ NOT ON SIZE ERROR statement-2 ]
```

```
        [ END-ADD ]
```

Format 2

```
ADD { CORRESPONDING } group-1 TO group-2 [ ROUNDED ]
    { CORR          }
    [ ON SIZE ERROR statement-1 ]
    [ NOT ON SIZE ERROR statement-2 ]
    [ END-ADD ]
```

Syntax

1. numeric-data-n is a numeric data item.
2. numeric-n is a literal or data item, or the output of an intrinsic function that is numeric.
3. group-n is a group data item containing one or more elementary data items.
4. statement-n is an imperative statement.

General Rules

1. In a FORMAT 1 ADD statement containing a GIVING clause, all numeric-n data items are added together, with the result stored in numeric-data-1.
2. In a FORMAT 1 ADD statement with no GIVING clause, all numeric-n data items are added, in sequence, to each of the data items following the TO clause.
3. The ROUNDED clause is applied when an arithmetic operation produces a result that includes more decimal places than are included in the description of the data item given to hold the final result of the arithmetic operation.
4. For rules regarding the ROUNDED clause, see the entry for ROUNDED in 5.3.1 Common General Rules.
5. The SIZE ERROR exception is triggered if the ON SIZE ERROR clause is present and if the receiving field is not large enough to accommodate the result of the ADD function.
6. For rules regaring the ON SIZE ERROR clause, see the entry for ON SIZE ERROR in 5.3.1 Common General Rules.
7. In an ADD CORRESPONDING operation, elementary items in separate group items must have the same elementary data name for the ADD function to be performed.
8. The ADD CORRESPONDING operation causes multiple ADD operations to be performed between elementary data items in separate group items, where the elementary items have the same elementary data name, and are not described as FILLER.
9. The rules for identifying a CORRESPONDING data item for the purposes of the ADD CORRESPONDING clause are the same as the rules used for the purposes of the MOVE CORRESPONDING clause.
10. For details on how data items are identified as CORRESPONDING, see 5.3.1 Common General Rules / Rules for identifying CORRESPONDING data elements.
11. In an ADD CORRESPONDING operation, all ADD operations will be completed before the ON SIZE ERROR condition will be triggered.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  ADD-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 FIELD-1 PIC 9V9 VALUE 2.5.
       77 FIELD-2 PIC 9V9 VALUE 3.4.
       77 FIELD-3 PIC 9.9.

       01 GROUP-1.
         03 FLD-1    PIC 99 VALUE 10.
         03 FLD-1    PIC 99 VALUE 20.
       01 GROUP-2.
         03 FLD-1    PIC 99 VALUE 5.
         03 FLD-2    PIC 99 VALUE 15.

       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
      *      ADD { INTEGER }  TO { INTEGER-LIST }
      *        [ GIVING INTEGER [ROUNDED] ]
      *        [ ON SIZE ERROR STATEMENT-1 ]
      *        [ NOT ON SIZE ERROR STATEMENT-2 ]
      *        [ END-ADD ]


            ADD FIELD-1 TO FIELD-2 GIVING FIELD-3
              ON SIZE ERROR DISPLAY "INVALID ADDITION" LINE 10 COL 10
              NOT ON SIZE ERROR DISPLAY FIELD-3 LINE 10 COL 10
            END-ADD.

      *      ADD LENGTH OF { INTEGER }  TO { INTEGER-LIST }
      *        [ GIVING INTEGER [ROUNDED] ]
      *        [ ON SIZE ERROR STATEMENT-1 ]
      *        [ NOT ON SIZE ERROR STATEMENT-2 ]
      *        [ END-ADD ]

            ADD LENGTH OF FIELD-1 TO FIELD-2 GIVING FIELD-3
              ON SIZE ERROR DISPLAY "INVALID ADDITION" LINE 11 COL 10
              NOT ON SIZE ERROR DISPLAY FIELD-3 LINE 11 COL 10
            END-ADD.

      *ADD {CORRESPONDING} GROUP-ITEM-1 TO GROUP-ITEM-2  [ ROUNDED ]
      *     { CORR          }
      *     [ ON SIZE ERROR STATEMENT-1 ]
      *     [ NOT ON SIZE ERROR STATEMENT-2 ]
      *     [ END-ADD ]

            ADD CORRESPONDING GROUP-1 TO GROUP-2
              ON SIZE ERROR DISPLAY "INVALID ADDITION" LINE 12 COL 10
              NOT ON SIZE ERROR
                DISPLAY FLD-1 OF GROUP-2 LINE 12 COL 10
                DISPLAY FLD-2 OF GROUP-2 LINE 13 COL 10
            END-ADD.

            DISPLAY "ADD1 FINISHED!" LINE 15 COL 10.
            ACCEPT DUMMY LINE 15 COL 30.
            STOP RUN.
```

5.3.4 ALLOCATE Statement

The ALLOCATE Statement allocates memory.

General Format

```
ALLOCATE [{numeric-data-1 CHARACTERS}
          {data-ptr-1                }
         [ INITIALIZED ]
         [ RETURNING data-ptr-2 ]
```

Syntax

1. numeric-data-1 is a numeric data item or literal with a positive integer value.
2. data-ptr-n is a data item declared with USAGE POINTER.

General Rules

1. if data-ptr-1 is specified, it requires the BASED clause.
2. When initialized, the address of the based data item referenced by data-ptr-1 is set to the predefined address NULL.
3. If data-ptr-1 is not specified, data-ptr-2 must be specified.
4. The allocated storage persists until explicitly released with a FREE statement or the run unit is terminated, whichever occurs first.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  ALLOCATE-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 DATA-PTR-1           USAGE POINTER BASED.
       77 DATA-PTR-2           USAGE POINTER.
       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
      *ALLOCATE [{EXPR CHARACTERS}[INITIALIZED][RETURNING DATA-PTR-2]
      *          {DATA-PTR-1     }

           ALLOCATE DATA-PTR-1 INITIALIZED.
           ALLOCATE 100 CHARACTERS INITIALIZED RETURNING DATA-PTR-2.

           DISPLAY "ALLOCATE-1 FINISHED!" LINE 10 COL 10.
           FREE DATA-PTR-1, DATA-PTR-2.
           ACCEPT DUMMY LINE 10 COL 30.
           STOP RUN.
```

5.3.5 ALTER Statement

The ALTER Statement provides a mechanism for re-directing the destination of the GO TO Statement.

General Format

```
ALTER { proc-1 TO PROCEED TO proc-2 } …
```

Syntax

1. proc-n is the name of a procedure or section in the program.

General Rules

1. The ALTER statement indicates that where proc-n is indicated as the target of a GO TO statement, the program should substitute an a different target proc.
2. GO TO statements targeting a proc-n that has been ALTER'ed are redirected to the proc named after the PROCEED TO clause in the ALTER statement.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  ALTER-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 ALTER-FLAG PIC 9 VALUE 1.
       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
           DISPLAY "IF ALTER-FLAG..." LINE 2 COL 10.

           IF ALTER-FLAG = 1
             DISPLAY "ALTER ALTER-1-EXIT.... " LINE 3 COL 10
             ALTER ALTER-1-EXIT TO PROCEED TO ALTER-2-EXIT
           END-IF.

           DISPLAY "GO TO ALTER-1-EXIT...." LINE 4 COL 10.
           GOTO ALTER-1-EXIT.


       ALTER-1-EXIT.
           GOTO LABEL-1.

       LABEL-1.
           DISPLAY "ALTER-1 EXIT!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.
           STOP RUN.
```

```
       ALTER-2-EXIT.
           DISPLAY "ALTER-2 EXIT!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.
           STOP RUN.
```

5.3.6 CALL Statement

The CALL Statement calls a sub-program, and passes parameters that are referenced in the LINKAGE SECTION of the CALLed program.

General Format

```
CALL program-identifier-1
[ USING { [ BY {REFERENCE} ] {data-1} ...} ...]
                {CONTENT }    {OMITTED }
                {VALUE }      {NULL }

[ {RETURNING} INTO identifier-2 ]
  {GIVING }

[ ON {EXCEPTION} statement-1 ]
     {OVERFLOW }

[ NOT ON {EXCEPTION} statement-2 ]
         {OVERFLOW }

[ END-CALL ]
```

Syntax

1. program-identifier-n is a program name, which may be expressed as a data element, literal, or data returned from a function call.
2. data-n is a data item.
3. identifier-n is a numeric or alphanumeric data item.
4. statement-n is an imperative statement.

General Rules

1. Program-identifier-n is the name of the subprogram to be called. The rules that COBOL-IT uses for resolving program names, and locating the named module are described in Getting Started with COBOL-IT Compiler Suite, in the Calling subprograms chapter.
2. The RETURNING data item may be numeric or alphanumeric. When CALL'ing a "C" program, which is designed to return a data item that is non-numeric/non-alphanumeric such as a pointer, the COBOL variable receiving the data item must be described with a PIC X, notation, or as a group item.
3. The GIVING/RETURNING phrase may precede the USING phrase.
4. For details about how the CALLing program will search for the target program being

CALL'ed, see Getting Started with COBOL-IT. Note that where target programs are entered in lower-case, the CALL statement will also search for the symbol in upper-case, and vica-versa.

The CALL'ed subprogram

1. If the CALL'ed subprogram contains the IS INITIAL PROGRAM clause, then the subprogram is placed into its INITIAL state every time it is CALL'ed.
2. If the CALL'ed subprogram does not contain the IS INITIAL PROGRAM clause, then the subprogram is placed into its INITIAL state the first time it is called. However, for all subsequent CALL's, the program retains its state in resident memory. Retained state includes the value of variables in LOCAL-STORAGE, and WORKING-STORAGE Sections, and the OPEN state of files, as well as file position.
3. If a subprogram that does not contain the IS INITIAL PROGRAM clause is the target of a CANCEL verb, then the next time it is CALL'ed, it is placed inot its INITIAL state.

The USING clause...

1. The USING clause names the parameters that are passed to the subprogram.
2. The manner in which the parameters are passed is determined by the BY clause, which indicates whether the parameters are passed BY REFERENCE (the default), BY VALUE, or BY CONTENT.
3. The order in which the parameters appear in the USING clause of the CALL'ing program must match the order in which the parameters are listed in the USING clause of the PROCEDURE DIVISION statement in the CALL'ed program.
4. The USING clause allows figurative constants to be used as parameters. In the case below, the figurative constant zero is considered to be alphanumeric data that is one byte in length.

Example:

```
CALL "MYPROG" USING ZERO.
```

The BY... clause

1. The BY REFERENCE clause indicates that the address of the data item has been passed through the Linkage Section. Updates to the data item are made directly to the memory address of the data item, and changes to the value of the data item will be seen in the CALL'ing program.
2. The BY VALUE clause indicates that the VALUE, rather than memory address of the item is passed through the Linkage Section. Updates to the data item will only be seen locally in the sub-program, and will not be returned through the data item to the CALL'ing program.
3. The BY CONTENT clause indicates that a copy of the address of the data item has been passed through the Linkage Section. Updates to the data item will only be seen locally in the sub-program, and will not be returned through the data item to the CALL'ing program.
4. BY REFERENCE **is** assumed as the default if no BY clause is present.
5. In the RETURNING/GIVING clause, RETURNING and GIVING are synonyms.

The ON EXCEPTION/ON OVERFLOW clause...

1. ON EXCEPTION and ON OVERFLOW are synonyms.
2. The EXCEPTION/OVERFLOW condition is triggered if the subprogram that is the target of the CALL statement cannot be loaded and executed.
3. If an EXCEPTION/OVERFLOW condition is triggered, and there is **no** ON EXCEPTION/ON OVERFLOW clause, the following occur:
    a. Program execution is halted,
    b. The error message "Cannot find module '[subprogram name]' is written to stderr.
    c. If the module has been compiled with –fsource-location, -debug, or –g, then the line number of the CALL will also be included in the output message, for example: C:/COBOL/CobolIT/Samples/call1.cbl:10: libcob: Cannot find module 'subpgm' Otherwise, the line number is represented as "0", for example: C:/COBOL/CobolIT/Samples/call1.cbl:0: libcob: Cannot find module 'subpgm'
    d. If the program has been compiled with –g or –fmem-info, the exception condition will also include a memory dump, for example:

```
C:\COBOL\CobolIT\Samples>cobc -g call1.cbl
C:\COBOL\CobolIT\Samples>cobcrun call1
C:/COBOL/CobolIT/Samples/call1.cbl:10: libcob: Cannot find module 'subpgm'


Cobol memory dump
+++++++++++++++++

PROGRAM ID : call1   (C:/COBOL/CobolIT/Samples/call1.cbl)
 Current line : C:/COBOL/CobolIT/Samples/call1.cbl:10
-----------------------------
         WORKING-STORAGE
                RETURN-CODE = +000000000
                TALLY = +000000000
                SORT-RETURN = +000000000
                NUMBER-OF-CALL-PARAMETERS = +000000000
                ws-dummy =
                COB-CRT-STATUS = 0000
-----------------------------
```

1. If an EXCEPTION/OVERFLOW condition is triggered, and there is an ON EXCEPTION/ON OVERFLOW clause, the following occur:
    a. Program execution is not halted.
    b. The imperative statement(s) contained within the ON EXCEPTION/ON OVERFLOW clause are executed.
    c. The NOT ON EXCEPTION/NOT ON OVERFLOW clause can be used to cause a set of imperative statements to be executed in the case where a CALL statement is performed, and returns without triggering an ON EXCEPTION condition.
    d. Statements associated with a NOT ON EXCEPTION/NOT ON OVERFLOW clause are executed after the CALL'ed program has finished executing, and returned control to the CALL'ing program.

LINKAGE-related runtime errors

1. In the case where the Linkage Section of a CALL'ed program contains more parameters than are passed by the CALL'ing program, a memory allocation error will result if the parameter not passed by the CALL'ing program is referenced in the CALL'ed program.
2. If a parameter passed by the CALL'ing program is smaller than the corresponding data item in the Linkage Section of the CALL'ed program, no error is generated. However, a memory allocation error can result if undefined memory is referenced in the CALL'ed program.
3. If a parameter passed by the CALL'ing program is larger than the corresponding data item in the Linkage Section of the CALL'ed program, no error is generated. The smaller receiving data item truncates the larger data item to be passed.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  CALL-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 DUMMY PIC X.
       77 SUBPGM-NAME  PIC X(5) VALUE "SUB-2".
       77 DATA-1       PIC X(5) VALUE "HELLO".
       77 RTN-VALUE    PIC 99 VALUE 0.
       PROCEDURE DIVISION.
           CALL "SUB-2" USING BY REFERENCE DATA-1
              RETURNING RTN-VALUE
              ON EXCEPTION
                DISPLAY "SUB-2 NOT FOUND!" LINE 10 COL 10
                ACCEPT DUMMY LINE 10 COL 30
                STOP RUN
           END-CALL.

           CALL"SUB-2".
           CALL SUBPGM-NAME.
           CALL "SUB-2" USING BY REFERENCE DATA-1.
           CALL "SUB-2" USING BY CONTENT DATA-1.
           CALL "SUB-2" USING BY VALUE DATA-1.


           DISPLAY "CALL-1 COMPLETE!" LINE 15 COL 10.
           ACCEPT DUMMY LINE 15 COL 30.
           STOP RUN.
```

The CALL Prototype

General Rules for the CALL prototype

1. ENTRY declarations must use parameters declared either as TYPEDEF, or with the reserved word ANY.
2. The external program must be properly structured. That is, it must contain a declaration for each of the IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, DATA DIVISION, and PROCEDURE DIVISION.
3. The DELIMITED clause is supported in the prototype definition,

For example:

```
entry C-FUNCTION-VAL
     c-call using By reference schar delimited
```

4. When the DELIMITED clause is used, strings passed to the “C” function are automatically null-terminated.
5. For an example, see the Code Samples below for prog.cbl, cproto.cpy, and cfunc.c.

Code Sample (prog.cbl)

```
COPY "cproto.cpy".
IDENTIFICATION   DIVISION.
PROGRAM-ID.      prog.
ENVIRONMENT      DIVISION.
CONFIGURATION SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01 grpstr.
   03 AV-STR PIC XX VALUE 'AV'.
   03 STR PIC X(10) VALUE "STR10".
   03 AP-STR PIC XX VALUE 'AP'.
   03 zz-STR PIC 0  VALUE X'00'.
01 A1  PIC 99 VALUE 1.
01 A2  PIC 99 VALUE 2.
01 A3  PIC 99 VALUE 3.
01 A4  PIC 99 VALUE 4.

PROCEDURE        DIVISION.

CALL "cfunction_val" USING "vallit" 1 2 3 4.
CALL "cfunction_ref" USING "reflit" 1 2 3 4.
DISPLAY "'" grpstr "'"
CALL "cfunction_val" USING STR A1 A2 A3 A4.
CALL "cfunction_ref" USING STR A1 A2 A3 A4.
DISPLAY "'" grpstr "'"
CALL "cfunction_any" USING grpstr function BYTE-LENGTH(grpstr).

.
```

Code Sample (cproto.cpy)

```
program-id. "c_typedefs" is external.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  schar                   pic x is typedef.
77  uns-schar               pic  9(2)  comp-5 is typedef.
77  short                  pic s9(4)  comp-5 is typedef.
77  uns-short              pic  9(4)  comp-5 is typedef.
77  int                    pic s9(9)  comp-5 is typedef.
```

```
 77  uns-int                pic  9(9)  comp-5 is typedef.
 77  long                   pic s9(9)  comp-5 is typedef.
 77  uns-long              pic  9(9)  comp-5 is typedef.
 77  l-long                pic s9(18) comp-5 is typedef.
 77  uns-l-long            pic  9(18) comp-5 is typedef.
 end program "c_typedefs".

 program-id. "c_typedefs" is external.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
special-names.
     call-convention 3 is Pascal
     call-convention 0 is c-call.
$set constant C-FUNCTION-VAL "cfunction_val"
$set constant C-FUNCTION-REF "cfunction_ref"
$set constant C-FUNCTION-ANY "cfunction_any"

 PROCEDURE DIVISION.
  entry
     C-FUNCTION-VAL
     c-call using
         by reference  schar delimited
         by value  uns-schar
         by value  uns-short
         by value  uns-int
         by value  uns-l-long
     returning         int

  entry
     C-FUNCTION-REF
     c-call using
         by reference  schar delimited
         by reference  uns-schar
         by reference  uns-short
         by reference  uns-int
         by reference  uns-l-long
     returning         int

  entry
     C-FUNCTION-ANY
     c-call using
         by reference  any
         by value      uns-int
     returning         int
  .
 end program "c_typedefs".
```

Code Sample (cfunc.c)

```
#include "stdio.h"

int cfunction_val(char *s, char c1, short c2, int c3, long long
c4)
{
    printf("'%s' %d %d %d %d %lld\n", s, strlen(s), (int)c1,
(int)c2, c3, c4);
}

int cfunction_ref(char *s, char *c1, short *c2, int *c3, long long
*c4)
{
    printf("'%s' %d ", s, strlen(s));
    if (c1) {
        printf("%d ",  (int)(*c1));
    } else {
        printf("<NULL>");
    }
    if (c2) {
        printf("%d ",  (int)(*c2));
    } else {
        printf("<NULL>");
    }
    if (c3) {
        printf("%d ", (int)(*c3));
    } else {
        printf("<NULL>");
    }
    if (c4) {
        printf("%lld\n",  (*c4));
    } else {
        printf("<NULL>\n");
    }
}

int cfunction_any(void *any , int c1)
{
    printf("'%s' %d\n", (long long)any, (int)c1);
}
```

5.3.7 CANCEL Statement

The CANCEL Statement causes a program to be returned to its initial state.

General Format

```
CANCEL { program-1 } ...
```

Syntax

1. program-n is a program name, which may be expressed as a data element, literal, or data returned from a function call.

General Rules

1. If a subprogram that does not contain the IS INITIAL PROGRAM clause is the target of a CANCEL verb, then the next time it is CALL'ed, it is placed into its INITIAL state.

   Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  CANCEL-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 DUMMY PIC X.
       77 SUBPGM-NAME  PIC X(5) VALUE "SUB-1".
       PROCEDURE DIVISION.
           CALL "SUB-1".
           CANCEL "SUB-1".
           DISPLAY "CALL/CANCEL [LITERAL]" LINE 7 COL 10.

           CALL SUBPGM-NAME.
           CANCEL SUBPGM-NAME.
           DISPLAY "CALL/CANCEL [VARIABLE]" LINE 8 COL 10.

           DISPLAY "CANCEL-1 COMPLETE!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.
           STOP RUN.
      *
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  SUB-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       PROCEDURE DIVISION.
           DISPLAY "SUB-1 COMPLETE!" LINE 5 COL 10.
           EXIT PROGRAM.
```

5.3.8 CHECKPOINT Statement

The CHECKPOINT Statement saves the current state of a program (field values, call stack, perform stack) into a file named <checkpoint prefix><program-id>.ctx.

Format 1

```
CHECKPOINT [checkpoint prefix] CONTINUE
[ { GIVING } numeric-data-1 ]
```

Format 2

```
     CHECKPOINT [checkpoint prefix] EXIT
                [ { RETURNING } numeric-1 ]
                  { WITH      }
```

Syntax

1. numeric-n is a literal or data item that is numeric.
2. Numeric-data-n is a numeric data item.

General Rules

1. When used with CONTINUE, the runtime saves the program state, and continues with execution.
2. When used with EXIT RETURNING [**numeric-data-1**], the runtime saves the program state, exits the program, and returns [**numeric-data-1**] to the calling program.
3. When running a program with a checkpoint file, the runtime reloads the program state from the checkpoint file, and continues execution at the statement following the CHECKPOINT line. If the GIVING [**numeric-data-1**] clause is specified, [**numeric-data-1**] is set to "1" when reloading and to "0" when execution resumes.
4. Use of the CHECKPOINT verb requires that all programs in the call stack be compiled with -fcheckpoint.
5. Reloading a program with a checkpoint file is done by using the –reload argument as follows:

   cobcrun --reload myprog

   Or

   cobcrun –-reload -–checkpoint <checkpoint file>myprog

5.3.9 CLOSE Statement

The CLOSE statement is used to CLOSE a file or files.

General Format

```
CLOSE { file-1 {REEL} [ FOR REMOVAL ] } ...
               {UNIT}
               [ WITH LOCK ]
               [ WITH NO REWIND ]
```

Syntax

1. file-n is a file described in the File Section with an FD.

General Rules

1. The CLOSE statement may only be applied to files that are in an OPEN state.
2. When a file is CLOSEd, all record and file locks held by the file are released.
3. The REEL, UNIT, FOR REMOVAL, and WITH NO REWIND clauses are checked for syntax, but are otherwise treated as commentary.
4. The REEL, UNIT, FOR REMOVAL, and WITH NO REWIND clauses can only be used with files described with ORGANIZATION SEQUENTIAL.
5. CLOSE file-1 WITH LOCK prevents file-1 from being OPEN'ed again in the same program instance.
6. The CLOSE operation updates the FILE STATUS variable.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CLOSE-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT TAPEFILE ASSIGN TO "TAPE-1"
           ORGANIZATION IS SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS TAPEFILE-STAT.
       DATADIVISION.
       FILE SECTION.
       FD TAPEFILE.
       01 TAPEFILE-RECORD          PIC X(80).
       WORKING-STORAGESECTION.
       77 TAPEFILE-STAT PIC XX.
       77 DUMMY                    PIC X.
       PROCEDURE DIVISION.
       MAIN.
      *CLOSE { FILE-NAME {REEL} [ FOR REMOVAL ] } ...
      *                  [UNIT}
      *                  [ WITH LOCK ]
      *                  [ WITH NO REWIND ]
            OPEN INPUT TAPEFILE.
            CLOSE TAPEFILE .

            OPEN INPUT TAPEFILE.
            CLOSE TAPEFILE WITH NO REWIND.

            OPEN INPUT TAPEFILE.
            CLOSE TAPEFILE WITH LOCK.

            OPEN INPUT TAPEFILE.
            CLOSE TAPEFILE REEL.

            OPEN INPUT TAPEFILE.
            CLOSE TAPEFILE REEL FOR REMOVAL.

            OPEN INPUT TAPEFILE.
```

```
           CLOSE TAPEFILE UNIT.

            OPEN INPUT TAPEFILE.
            CLOSE TAPEFILE UNIT FOR REMOVAL.

            DISPLAY "CLOSE-1 COMPLETE!" LINE 10 COL 10.
            ACCEPT DUMMY LINE 10 COL 30.
            STOP RUN.
```

5.3.10 COMMIT Statement

The COMMIT statement unlocks locked records and flushes buffers to disk.

General Format

```
COMMIT.
```

General Rules

1. The COMMIT statement causes unwritten file buffers to be written to the target file.
2. After writing file buffers, the COMMIT statement releases record and file locks held by the target file.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. COMMIT-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT RESWORDS ASSIGN TO "RESWORDS"
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS RESERVED-WORD
           FILE STATUS IS RESWORDS-STAT.

       DATA DIVISION.
       FILE SECTION.
       FD RESWORDS.
       01 RESWORDS-RECORD.
         03 RESERVED-WORD         PIC X(30).

       WORKING-STORAGE SECTION.
       77 RESWORDS-STAT PIC XX.
         88 END-OF-RESWORDS  VALUE "10".
       77 DUMMY                  PIC X.

       PROCEDURE DIVISION.
       MAIN.
           OPEN OUTPUT RESWORDS.
           MOVE "ACCEPT" TO RESERVED-WORD.
```

```
           WRITE RESWORDS-RECORD.
           COMMIT.
           DISPLAY "COMMIT-1 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.

           CLOSE RESWORDS.
           STOP RUN.
```

5.3.11 COMPUTE Statement

The COMPUTE statement performs arithmetic computations on multiple operands and stores the results.

General Format

```
COMPUTE { numeric-data-1 [ROUNDED] } ... { =     } arithmetic-expr
                                         { EQUAL }
        [ ON SIZE ERROR statement-1 ]
        [ NOT ON SIZE ERROR statement-2 ]
        [ END-COMPUTE ]
```

Syntax

1. numeric-data-n is a numeric data item.
2. statement-n is an imperative statement.

General Rules

1. EQUAL and "=" are synonyms.
2. The ROUNDED clause is applied when an arithmetic operation produces a result that includes more decimal places than are included in the description of the data item given to hold the final result of the arithmetic operation.
3. For rules regarding the ROUNDED clause, see the entry for ROUNDED in 5.3.1 Common General Rules.
4. The SIZE ERROR exception is triggered if the ON SIZE ERROR clause is present and if the receiving field is not large enough to accommodate the result of the arithmetic operation.
5. For rules regarsding the ON SIZE ERROR clause, see the entry for ON SIZE ERROR in 5.3.1 Common General Rules.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. COMPUTE-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
```

```
       77 FLD-1                    PIC Z(9).9(9).
       77 OP-1                     PIC 9 VALUE 1.
       77 OP-2                     PIC 9 VALUE 7.
       77 OP-3                     PIC 9 VALUE 4.
       77 TEMP-STRING              PIC X(40).
       77 DUMMY                    PIC X.

       PROCEDURE DIVISION.
       MAIN.
      *COMPUTE { INT-LIST [ROUNDED] } ... { =     } ARITHMETIC-EXPR
      *                                  { EQUAL }
      *        [ ON SIZE ERROR STATEMENT ]
      *        [ NOT ON SIZE ERROR STATEMENT ]
      *        [ END-COMPUTE ]
      *
           COMPUTE FLD-1 ROUNDED = ( OP-1 / OP-2 ) * OP-3
             ON SIZE ERROR
               DISPLAY "SIZE ERROR!" LINE 12 COL 10
               PERFORM EXIT-COMPUTE
             NOT ON SIZE ERROR
               CONTINUE
           END-COMPUTE.

           PERFORM DISPLAY-RESULTS.
           PERFORM EXIT-COMPUTE.
           DISPLAY-RESULTS.
           DISPLAY FLD-1 LINE 5, COL 10.
           STRING"(", DELIMITED BY SIZE,
                      OP-1, DELIMITED BY SIZE,
                       " / ", DELIMITED BY SIZE,
                       OP-2,  DELIMITED BY SIZE,
                       ")",   DELIMITED BY SIZE,
                       " * ", DELIMITED BY SIZE,
                       OP-3,  DELIMITED BY SIZE,
                       " = ", DELIMITED BY SIZE,
                       FLD-1, DELIMITED BY SIZE,
                       INTO TEMP-STRING.

           DISPLAY TEMP-STRING LINE10, COL10.
       EXIT-COMPUTE.
           DISPLAY "COMPUTE-1 COMPLETE!" LINE 12 COL 10.
           ACCEPT DUMMY LINE 12 COL 30.
           STOP RUN.
```

5.3.12 CONTINUE Statement

The CONTINUE statement prescribes no action.

General Format

```
CONTINUE
```

General Rules

1. The CONTINUE statement is an imperative statement that prescribes no action.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CONTINUE-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 FLD-1                    PIC 9 VALUE 1.
       77 FLD-2                    PIC 9 VALUE 2.
       77 FLD-3                    PIC 9.
       77 DUMMY                    PIC X.

       PROCEDURE DIVISION.
       MAIN.
      *
      *CONTINUE
      *
            ADD FLD-1 TO FLD-2 GIVING FLD-3
              ON SIZE ERROR DISPLAY "SIZE ERROR" LINE 8 COL 10
              NOT ON SIZE ERROR CONTINUE
            END-ADD.

            DISPLAY "CONTINUE-1 COMPLETE!" LINE 10 COL 10.
            ACCEPT DUMMY LINE 10 COL 30.
            STOP RUN.
```

5.3.13 DELETE Statement

The DELETE statement logically removes a record from a relative, or indexed file.

General Format

Format 1

```
DELETE file-1 RECORD
     [ INVALID KEY statement-1 ]
     [ NOT INVALID KEY statement-2 ]
     [ END-DELETE ]
```

Syntax

1. file-n is a file described in the File Section with an FD.

General Rules

1. file-1 must be described with ORGANIZATION IS RELATIVE or ORGANIZATION IS INDEXED.
2. file-1 must be OPEN I-O when the DELETE file-1 RECORD statement is executed.
3. For files declared with ACCESS IS SEQUENTIAL
    a. The INVALID KEY/NOT INVALID KEY clauses cannot be used.

   b. The previous I/O statement must be a successful READ. That record is DELETE'd from the file.
4. For files declared with ACCESS IS DYNAMIC or ACCESS IS RANDOM
   a. The INVALID KEY/NOT INVALID KEY clauses can be used
5. For relative files, the RELATIVE KEY is used to identify the record to be DELETE'd. If the RELATIVE KEY data item does not identify a record, then the INVALID KEY condition is raised.
6. For indexed files, the RECORD KEY is used to identify the record to be DELETE'd. If the RECORD KEY data item does not identify a record, then the INVALID KEY condition is raised.
7. The INVALID KEY condition can be handled programmatically using the INVALID KEY clause. Statement-1 executes when the INVALID KEY condition is detected. Statement-2 executes when the INVALID KEY condition is NOT detected.
8. In the absence of the INVALID KEY clause, the INVALID KEY condition causes a File I/O error condition, which can be handled in DECLARATIVES. If it is not handled in DECLARATIVES, or if there are no DECLARATIVES, then the INVALID KEY condition updates the file-status variable and aborts the program.
9. If successful, the DELETE file-1 RECORD statement removes the identified record from the file.
10. The DELETE file-1 RECORD statement updates the FILE-STATUS variable.

Format 2

DELETE FILE <filename> erases the file and the optional index.

```
DELETE FILE file-1
      [ END-DELETE ]
```

Syntax

1. file-n is a file described in the File Section with an FD.

General Rules

1. When executing a DELETE FILE file-1 statement, file-1 must be CLOSEd.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DELETE-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT RESWORDS ASSIGN TO "RESWORDS"
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS RESERVED-WORD
           FILE STATUS IS RESWORDS-STAT.
```

```
       DATA DIVISION.
       FILE SECTION.
       FD RESWORDS.
       01 RESWORDS-RECORD.
         03 RESERVED-WORD         PIC X(30).

       WORKING-STORAGE SECTION.
       77 RESWORDS-STAT PIC XX.
         88 END-OF-RESWORDS  VALUE "10".
       77 DUMMY                  PIC X.

       PROCEDURE DIVISION.
       MAIN.
           OPEN OUTPUT RESWORDS.
           MOVE "ACCEPT" TO RESERVED-WORD.
           WRITE RESWORDS-RECORD.
           MOVE "ADD" TO RESERVED-WORD.
           WRITE RESWORDS-RECORD.
           CLOSE RESWORDS.

           OPEN I-O RESWORDS.
           MOVE "ACCEPT" TO RESERVED-WORD.
           READ RESWORDS.

      * DELETE FILE-NAME RECORD
      *    [ INVALID KEY STATEMENT-1 ]
      *    [ NOT INVALID KEY STATEMENT-2 ]
      *    [ END-DELETE ]

           DELETE RESWORDS RECORD.
           IF RESWORDS-STAT = "00"
             DISPLAY"RECORD DELETED!"LINE 5 COL 10
           ELSE
             DISPLAY "ERROR! FILE STATUS: " LINE 5 COL 10
             DISPLAY RESWORDS-STAT LINE 5 COL 31
           END-IF.

           CLOSE RESWORDS.
      *
      * DELETE FILE FILE-NAME
      *    [ END-DELETE ]

           DELETE FILE RESWORDS.
           IF RESWORDS-STAT = "00"
             DISPLAY"FILE DELETED!"LINE 7 COL 10
           ELSE
             DISPLAY "ERROR! FILE STATUS: " LINE 7 COL 10
             DISPLAY RESWORDS-STAT LINE 7 COL 31
           END-IF.

           DISPLAY "DELETE-1 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.

           STOP RUN.
```

5.3.14 DISPLAY Statement

The DISPLAY statement causes data stored in an identifier (literal, data item, or output from a function call) to be displayed on the terminal screen at the specified location, with the specified attributes.

Format 1

```
DISPLAY {identifier-1} … UPON mnemonic-name-1
 [WITH NO ADVANCING]
[END-DISPLAY]
```

Syntax

1. identifier-n is a data element, literal, or data returned from a function call.
2. mnemonic-name-1 is a system name associated with a hardware device, or a user name associated with a system name in the Special Names area.

General Rules

1. mnemonic-name-1 must be CONSOLE, CRT, PRINTER, or a user-defined name associated with CONSOLE, CRT, or PRINTER in SPECIAL-NAMES.
2. All mnemonic-name-1 are interpreted as CONSOLE.
3. In the absence of the UPON clause UPON CONSOLE is applied as the default.
4. WITH NO ADVANCING removes carriage return/line feed sequences from the end of the DISPLAY.

Format 2

```
DISPLAY {identifier-1} … UPON {ARGUMENT-NUMBER  }
                              {COMMAND-LINE     }
                              {ENVIRONMENT-NAME }
                              {ENVIRONMENT-VALUE}
                              {PRINTER}
[END-DISPLAY]
```

Syntax

1. identifier-n is a data element, literal, or data returned from a function call.

General Rules

1. DISPLAY identifier-1 UPON ARGUMENT NUMBER is recognized syntax, but is otherwise treated as commentary. A subsequent ACCEPT FROM ARGUMENT-NUMBER does not return the value of identifier-1.
2. DISPLAY identifier-1 UPON COMMAND-LINE modifies the COMMAND-LINE.
3. A subsequent ACCEPT FROM COMMAND-LINE statement returns the value of identifier-1.
4. Creating an environment variable, and assigning it a value can be done with the DISPLAY

verb, as follows:

a. DISPLAY identifier-1 UPON ENVIRONMENT-NAME
b. DISPLAY identifier-2 UPON ENVIRONMENT-VALUE
c. This sequence of statements is the equivalent of the SET ENVIRONMENT "ENVIRONMENT-NAME" TO "ENVIRONMENT-VALUE" statement.

5. The value of identifier-2 can be retrieved from the environment variable named identifier-1 using the ACCEPT FROM ENVIRONMENT statement.
6. DISPLAY identifier-1 UPON PRINTER, when used with a file defined with the COB_DISPLAY PRINTER environment variable, performs OPEN and WRITE statements on the file, and the file is CLOSEd after each DISPLAY.

Format 3

```
DISPLAY identifier-1
[AT {[LINE NUMBER    {identifier-2}]}]
     {COLUMN} NUMBER {identifier-3}
     {COL    }
     {identifier-4}
[ [WITH] NO ADVANCING ]
         [ [WITH] {BELL}       ]
                  {BEEP}
         [ [WITH] {BLINKING}   ]
                  {BLINK   }
         [ {HIGHLIGHT}         ]
           {LOWLIGHT }
         [ REVERSE-VIDEO       ]
         [ UNDERLINE           ]
         [ OVERLINE            ]
         [ FOREGROUND-COLOR is numeric-1 ]
         [ BACKGROUND-COLOR is numeric-2 ]
         [ SCROLL {UP  } numeric-3 LINES ]
                  {DOWN}
         [ BLANK LINE          ]
         [ BLANK SCREEN        ]
[END-DISPLAY]
```

Syntax

1. identifier-n is a data element, literal, or data returned from a function call.
2. identifier-4 is a 4-byte numeric value where the first two bytes represent the line number and the second 2 bytes represent the column number. This provides an alternative way to position a DISPLAY. In effect, DISPLAY [identifier-1] LINE 10 COLUMN 10 is synonymous with DISPLAY [identifier-1] AT 1010.
3. numeric-n is a literal or data item that is numeric.

General Rules

1. A field-level DISPLAY is positioned with the use of an AT statement, which is used to designate the line number and the column number. This can be done either through the separate use of the keywords LINE and COLUMN, or with a single 4-byte numeric value

that represents the location of the DISPLAY with a Line-Column designation, where bytes 1-2 represent the line number, and bytes 3-4 represent the column number.

2. DISPLAY field1 LINE 1 COLUMN 3 Field2 AUTO UPDATE
   Prior to this enhancement, each field had to be the target of a separate DISPLAY statement, and the AUTO UPDATE clause was supported as WITH AUTO UPDATE.
3. For information regarding the DISPLAY ATTRIBUTES, see **4.9.3 Screen Attributes**.

Format 4

```
DISPLAY screen-name-1
[AT {[LINE NUMBER    {identifier-2}]}]
     {COLUMN} NUMBER {identifier-3}
     {COL    }
     {identifier-4}
[END-DISPLAY]
```

Syntax

1. screen-name-n is the name of a screen element described in the Screen Section.
2. identifier-n is a data element, literal, or data returned from a function call.
3. identifier-4 is a 4-byte numeric value where the first two bytes represent the line number and the second 2 bytes represent the column number. This provides an alternative way to position a DISPLAY. In effect, DISPLAY [screen] LINE 10 COLUMN 10 is synonymous with DISPLAY [screen] AT 1010.

General Rules

1. For information regarding the DISPLAY ATTRIBUTES, see **4.9.3 Screen Attributes**.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DISPLAY-3.

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.

      *   THE SPECIAL-NAMES PARAGRAPH THAT FOLLOWS PROVIDES FOR THE
      *   CAPTURING OF THE F10 FUNCTION KEYAND FOR POSITIONING OF THE
      *   CURSOR.

       SPECIAL-NAMES.
           CURSOR IS CURSOR-POSITION
           CRT STATUS IS CRT-STATUS.

       DATA DIVISION.
       WORKING-STORAGE SECTION.

      *   CURSOR-LINE SPECIFIES THE LINE AND CURSOR-COL SPECIFIES THE
      *   COLUMN OF THE CURSOR POSITION.

       01  CURSOR-POSITION.
         02  CURSOR-LINE PIC 99.
```

```
          02  CURSOR-COL  PIC 99.

      *   NORMAL TERMINATION OF THE ACCEPT STATEMENT WILL RESULT IN A VALUE
      *   OF '0' IN KEY1.  WHEN THE USER PRESSES F10, THE VALUE IN KEY1 WILL
      *   BE '1' AND FKEY-10 WILL BE TRUE.

       01 CRT-STATUS.
          03 KEY1            PIC X.
          03 KEY2            PIC X.
            88 FKEY-10      VALUE11.
          03 FILLER PIC XX.

      *   THE FOLLOWING DATA ITEMS ARE FOR A "DAILY CALENDAR."  IT SHOWS
      *   THE DAY'S APPOINTMENTS AND ALLOWS APPOINTMENTS TO BE MADE,
      *   CANCELED, AND PRINTED.
       01 DATA-STORE.
          03 ACCEPT-ITEM1  PIC X.
          03 APPT-NAME     PIC X(80).
          03 APPT-DESC     PIC X(80).
          03 APPT-DATE.
            05 APPT-DAY PIC 99 COMP.
            05 APPT-MONTH PIC 99 COMP.
            05 APPT-YEAR  PIC 99 COMP.
          03 APPT-TIME.
            05 APPT-HOUR   PIC 99.
            05 APPT-MINUTE   PIC 99.
            05 APPT-MERIDIEM PIC XX.
          03 APPT-VERIFY   PIC X.
          03 EMPTY-LINEPIC X(80).

      *   THE SCREEN SECTION DESIGNS THE DAILY CALENDAR, WITH A MENU
      *   SCREEN FROM WHICH THE USER SELECTS AN OPTION:  TO SHOW
      *   APPOINTMENTS, SCHEDULE AN APPOINTMENT, CANCEL AN APPOINTMENT,
      *   AND PRINT THE APPOINTMENTS.

       SCREEN SECTION.

       01 MENU-SCREEN BLANK SCREEN
            FOREGROUND-COLOR7BACKGROUND-COLOR1.
          02 MENU-SCREEN-2.
            03 TITLE-BAR
                 FOREGROUND-COLOR7BACKGROUND-COLOR4.
              04 LINE1 PIC X(80) FROM EMPTY-LINE.
              04 LINE1 COLUMN 32 VALUE "DAILY CALENDAR".

            03 LINE 7 COLUMN 26
               PIC X TO ACCEPT-ITEM1.
            03 VALUE " SHOW APPOINTMENTS FOR A DAY ".
            03 LINE 9 COLUMN 26
               PIC X TO ACCEPT-ITEM1.
            03 VALUE " SCHEDULE AN APPOINTMENT ".
            03 LINE 11 COLUMN 26
               PIC X TO ACCEPT-ITEM1.
            03 VALUE " CANCEL AN APPOINTMENT ".
            03 LINE 13 COLUMN 26
               PIC X TO ACCEPT-ITEM1.
            03 VALUE " PRINT YOUR APPOINTMENTS ".
            03 HELP-TEXT
                 FOREGROUND-COLOR 6 BACKGROUND-COLOR 0.
              04 LINE 19 COLUMN 12
                 VALUE
             " USE THE ARROW KEYS TO MOVE THE CURSOR AMONG MENU ITEMS. ".
              04 LINE 20 COLUMN 12
```

```
             VALUE
          " PRESS <RETURN> WHEN THE CURSOR IS AT THE DESIRED ITEM.  ".
           04 LINE 21 COLUMN 12
             VALUE
          " PRESS <F10> TO EXIT.                                    ".

       01 SCHEDULE-SCREEN BLANK SCREEN.
         02 TITLE-BAR
              FOREGROUND-COLOR 7 BACKGROUND-COLOR 4.
           03  LINE1 PIC X(80) FROM EMPTY-LINE.
           03  LINE1 COLUMN30 VALUE "SCHEDULE APPOINTMENT".

         02 FIELDS-TEXT
              FOREGROUND-COLOR 7 BACKGROUND-COLOR 1.
           03 LINE 5 VALUE " DESCRIPTION OF APPOINTMENT: ".
           03 LINE PLUS 4 VALUE " DATE OF APPOINTMENT (DD/MM/YY): ".
           03 COLUMN PLUS 5 VALUE"/  /".
           03 LINE PLUS 2 VALUE " TIME OF APPOINTMENT (HH:MM MM): ".
           03 COLUMN PLUS 5 VALUE ":".

         02 FIELDS-INPUT
              FOREGROUND-COLOR 7 BACKGROUND-COLOR 0 AUTO.
           03 LINE 6 PIC X(80) TO APPT-NAME REQUIRED.
           03 LINE 7 PIC X(80) TO APPT-DESC.
           03 LINE 9 COLUMN 36 PIC 99 USING APPT-DAYFULL .
           03 LINE 9 COLUMN 39 PIC 99 USING APPT-MONTH FULL.
           03 LINE 9 COLUMN 42 PIC 99 USING APPT-YEAR.
           03 LINE 11 COLUMN 36 PIC 99 USING APPT-HOUR.
           03 LINE 11 COLUMN 39 PIC 99 USING APPT-MINUTE.
           03 LINE 11 COLUMN 42 PIC XX USING APPT-MERIDIEM.

         02 HELP-TEXT
            FOREGROUND-COLOR 6 BACKGROUND-COLOR 0.
           03 LINE 16 COLUMN 18
              VALUE " USE CURSOR KEYS TO MOVE WITHIN THE FIELDS. ".
           03 LINE 17 COLUMN 18
              VALUE " PRESS <TAB> TO ENTER NEXT FIELD.           ".
           03 LINE 18 COLUMN 18
              VALUE" PRESS <RETURN> WHEN FINISHED.              ".

       01 VERIFY-SUBSCREEN FOREGROUND-COLOR 7 BACKGROUND-COLOR 1.
         02 LINE 16 COLUMN 1 ERASE EOS.
         02 LINE 17 COLUMN 25 VALUE " IS THIS ENTRY CORRECT? (Y/N): ".
         02 PIC X USING APPT-VERIFY AUTO.

       PROCEDURE DIVISION.
       P0.

           DISPLAY MENU-SCREEN.

      *    THE CURSOR POSITION IS NOT WITHIN AN ITEM ON THE SCREEN, SO THE
      *    FIRST ITEM IN THE MENU WILL BE ACCEPTED FIRST.

           MOVE 0 TO CURSOR-LINE, CURSOR-COL.

      *    THE USER MOVES THE CURSOR WITH THE ARROW KEYS TO THE
      *    DESIRED MENU ITEM (TO SHOW, SCHEDULE, CANCEL, OR PRINT
      *    APPOINTMENTS) AND SELECTS THE ITEM BY PRESSING <RETURN>.
      *    IF THE USER WISHES TO EXIT WITHOUT SELECTING AN OPTION,
      *    THE USER CAN PRESS THE F10 FUNCTION KEY.

           ACCEPT MENU-SCREEN.
```

```
           IF KEY1 EQUAL "0"
             PERFORM OPTION_CHOSEN

           ELSE IF KEY1 EQUAL "1" AND FKEY-10
             DISPLAY "YOU PRESSED THE F10 KEY; EXITING..." LINE22.

           STOP RUN.


       OPTION_CHOSEN.

      *    FOR BREVITY, THE SAMPLE PROGRAM INCLUDES COMPLETE CODE
      *    FOR THE "SCHEDULE APPOINTMENT" SCREEN ONLY.  A COMPLETE
      *    PROGRAM FOR A CALENDAR WOULD ALSO INCLUDE CODE FOR
      *    DISPLAYING, CANCELING, AND PRINTING THE DAY'S APPOINTMENTS.

           IF CURSOR-LINE = 7
             DISPLAY "YOU SELECTED SHOW APPOINTMENTS" LINE 22.

           IF CURSOR-LINE = 9
             MOVE "01" TO APPT-DAY
             MOVE "01" TO APPT-MONTH
             MOVE "94" TO APPT-YEAR
             MOVE "12" TO APPT-HOUR
             MOVE "00" TO APPT-MINUTE
             MOVE "AM" TO APPT-MERIDIEM
             DISPLAY SCHEDULE-SCREEN

      *    THE USER TYPES THE DESCRIPTION, DATE, AND TIME OF THE
      *    APPOINTMENT.

            ACCEPT SCHEDULE-SCREEN

            MOVE "Y" TO APPT-VERIFY
            DISPLAY VERIFY-SUBSCREEN

      *    THE USER IS ASKED, "IS THIS ENTRY CORRECT?"  ANSWER IS
      *    Y OR N.

            ACCEPT VERIFY-SUBSCREEN.

            IF CURSOR-LINE = 11
            DISPLAY "YOU SELECTED CANCEL APPOINTMENTS" LINE 22.

            IF CURSOR-LINE = 13
            DISPLAY "YOU SELECTED PRINT APPOINTMENTS" LINE 22.

            ENDPROGRAM SCREEN2.
```

5.3.15 DIVIDE Statement

The DIVIDE statement performs arithmetic division, allowing for the storage of the quotient, and the remainder.

Format 1
The Format 1 DIVIDE Statement DIVIDEs a numeric-data-1 INTO a numeric-2 data item, or multiple numeric-2 data items, and moves the result of the DIVIDE operation into the

numeric-2 (or multiple numeric-2) data items. The result may be optionally ROUNDED. Remainder is not stored.

```
DIVIDE numeric-data-1 INTO { numeric-2 [ROUNDED] } ...
   [ ON SIZE ERROR statement-1 ]
   [ NOT ON SIZE ERROR statement-2 ]
   [ END-DIVIDE ]
```

Format 2

The Format 2 DIVIDE Statement DIVIDEs a numeric-data-1 INTO a numeric-data-2, and moves the result of the DIVIDE operation one or more numeric-3 data items, named as the target(s) of a GIVING clause. The result may be optionally ROUNDED. Remainder is not stored.

```
DIVIDE numeric-data-1 INTO numeric-data-2
          GIVING { numeric-3 [ROUNDED] } ...
    [ ON SIZE ERROR statement-1 ]
    [ NOT ON SIZE ERROR statement-2 ]
    [ END-DIVIDE ]
```

Format 3

The Format 3 DIVIDE Statement DIVIDEs a numeric-data-1 BY a numeric-data-2, and moves the result of the DIVIDE operation one or more numeric-3 data items, named as the target(s) of a GIVING clause. The result may be optionally ROUNDED. Remainder is not stored.

```
DIVIDE numeric-data-1 BY numeric-data-2
          GIVING { numeric-3 [ROUNDED] } ...
    [ ON SIZE ERROR statement-1 ]
    [ NOT ON SIZE ERROR statement-2 ]
    [ END-DIVIDE ]
```

Format 4

The Format 4 DIVIDE Statement DIVIDEs a numeric-data-1 INTO a numeric-data-2, and moves the result of the DIVIDE a numeric-3 data item, named as the target of a GIVING clause. The result may be optionally ROUNDED. Remainder is stored in a numeric-4 data item.

```
DIVIDE numeric-data-1 INTO numeric-data-2
          GIVING numeric-3 [ROUNDED]
```

```
            REMAINDER numeric-4
        [ ON SIZE ERROR statement-1 ]
        [ NOT ON SIZE ERROR statement-2 ]
        [ END-DIVIDE ]
```

Format 5

The Format 5 DIVIDE Statement DIVIDEs a numeric-data-1 item BY a numeric-data-2 item , and moves the result of the DIVIDE operation into a numeric-3 data item, named as the target of a GIVING clause. The result may be optionally ROUNDED. Remainder is stored in a numer-4 data item.

```
DIVIDE numeric-data-1 BY numeric-data-2
            GIVING numeric-3 [ROUNDED]
            REMAINDER numeric-4
        [ ON SIZE ERROR statement-1 ]
        [ NOT ON SIZE ERROR statement-2 ]
        [ END-DIVIDE ]
```

Syntax

1. numeric-n is a literal or data item that is numeric.
2. numeric-data-n is a numeric data item.
3. statement-n is an imperative statement.

General Rules

1. Format 1, Format 2, and Format 3 of the DIVIDE operation allow multiple receiving fields, but do not allow for the storing of the REMAINDER.
2. Format 4 and Format 5 of the DIVIDE operation only allow a single receiving field, but also allow for the storing of the REMAINDER.
3. The ROUNDED clause is applied when a DIVIDE operation produces a result that includes more decimal places than are included in the description of the data item given to hold the final result of the arithmetic operation.
4. For rules regarding the ROUNDED clause, see the entry for ROUNDED in 5.3.1 Common General Rules.
5. The SIZE ERROR exception is triggered if the ON SIZE ERROR clause is present and if the receiving field is not large enough to accommodate the result of the DIVIDE function.
6. The SIZE ERROR exception is triggered if numeric-data-1 has a value of zero.
7. For rules regaring the ON SIZE ERROR clause, see the entry for ON SIZE ERROR in 5.3.1 Common General Rules.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  DIVIDE-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       78 CONST-1           VALUE 10.
```

```
       78 CONST-2           VALUE 25.
       77 NUMERIC-1         PIC 99.
       77 NUMERIC-2         PIC 99.
       77 NUMERIC-3         PIC 9V99.
       77 NUMERIC-4         PIC 9V99.
       77 DUMMY             PIC X.
       PROCEDURE DIVISION.
       MAIN.
      *FORMAT 1
      *DIVIDE NUMERIC-1 INTO { NUMERIC-2 [ROUNDED] } ...
      * [ ON SIZE ERROR STATEMENT-1 ]
      * [ NOT ON SIZE ERROR STATEMENT-2 ]
      * [ END-DIVIDE ]

           MOVE CONST-1 TO NUMERIC-1.
           MOVE CONST-2 TO NUMERIC-2.

           DIVIDE NUMERIC-1 INTO NUMERIC-2 ROUNDED
             ON SIZE ERROR
               DISPLAY "FORMAT 1 SIZE ERROR!" LINE 3 COL 10
             NOT ON SIZE ERROR
               DISPLAY NUMERIC-2 LINE 3 COL 10
           END-DIVIDE.

      *FORMAT 2
      * DIVIDE NUMERIC-1 INTO NUMERIC-2
      *         GIVING { NUMERIC-3 [ROUNDED] } ...
      *  [ ON SIZE ERROR STATEMENT-1 ]
      *  [ NOT ON SIZE ERROR STATEMENT-2 ]
      *  [ END-DIVIDE ]
      *
           MOVE CONST-1 TO NUMERIC-1.
           MOVE CONST-2 TO NUMERIC-2.

           DIVIDE NUMERIC-1 INTO NUMERIC-2
             GIVING NUMERIC-3 ROUNDED
             ON SIZE ERROR
               DISPLAY "FORMAT 2 SIZE ERROR!" LINE 4 COL 10
             NOT ON SIZE ERROR
               DISPLAY NUMERIC-3 LINE 4 COL 10
           END-DIVIDE.


      *FORMAT 3
      * DIVIDE NUMERIC-1 BY NUMERIC-2
      *         GIVING { NUMERIC-3 [ROUNDED] } ...
      *   [ ON SIZE ERROR STATEMENT-1 ]
      *   [ NOT ON SIZE ERROR STATEMENT-2 ]
      *   [ END-DIVIDE ]
      *
           MOVE CONST-1 TO NUMERIC-1.
           MOVE CONST-2 TO NUMERIC-2.

           DIVIDE NUMERIC-1 BY NUMERIC-2
             GIVING NUMERIC-3 ROUNDED
             ON SIZE ERROR
               DISPLAY "FORMAT 3 SIZE ERROR!" LINE 5 COL 10
             NOT ON SIZE ERROR
               DISPLAY NUMERIC-3 LINE 5 COL 10
           END-DIVIDE.

      *FORMAT 4
      * DIVIDE NUMERIC-1 INTO NUMERIC-2
```

```
      *         GIVING NUMERIC-3 [ROUNDED]
      *         REMAINDER NUMERIC-4
      *    [ ON SIZE ERROR STATEMENT-1 ]
      *    [ NOT ON SIZE ERROR STATEMENT-2 ]
      *    [ END-DIVIDE ]
      *
           MOVE CONST-1 TO NUMERIC-1.
           MOVE CONST-2 TO NUMERIC-2.

           DIVIDE NUMERIC-1 INTO NUMERIC-2
             GIVING NUMERIC-3 ROUNDED
             REMAINDER NUMERIC-4
             ON SIZE ERROR
               DISPLAY "FORMAT 4 SIZE ERROR!" LINE 6 COL 10
             NOT ON SIZE ERROR
               DISPLAY NUMERIC-3 LINE 6 COL 10
               DISPLAY "REMAINDER" LINE 6 COL 20
               DISPLAY NUMERIC-4 LINE 6 COL 30
           END-DIVIDE.

      *FORMAT 5
      * DIVIDE NUMERIC-1 BY NUMERIC-2
      *         GIVING NUMERIC-3 [ROUNDED]
      *         REMAINDER NUMERIC-4
      *    [ ON SIZE ERROR STATEMENT-1 ]
      *    [ NOT ON SIZE ERROR STATEMENT-2 ]
      *    [ END-DIVIDE ]

           MOVE CONST-1 TO NUMERIC-1.
           MOVE CONST-2 TO NUMERIC-2.

           DIVIDE NUMERIC-1 BY NUMERIC-2
             GIVING NUMERIC-3 ROUNDED
             REMAINDER NUMERIC-4
             ON SIZE ERROR
               DISPLAY "FORMAT 5 SIZE ERROR!" LINE 7 COL 10
             NOT ON SIZE ERROR
               DISPLAY NUMERIC-3 LINE 7 COL 10
               DISPLAY "REMAINDER" LINE 7 COL 20
               DISPLAY NUMERIC-4 LINE 7 COL 30
           END-DIVIDE.

           DISPLAY "DIVIDE-1 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.

           STOP RUN.
```

5.3.16 ENTRY Statement

The ENTRY statement is used to establish an alternate entry point in the program.

General Format

```
ENTRY entry-name [ USING     { BY REFERENCE }   {data-1} ... ]
                             { BY VALUE     }
                             { BY CONTENT   }
```

Syntax

1. entry-name is a literal.
2. Name resolution for ENTRY routines is case-sensitive.
3. data-n is a data item.

General Rules

1. ENTRY statements cannot be used in nested subprograms.
2. The program containing the ENTRY statement must be loaded in memory, when CALL'ed.
3. The General Rules governing the USING clause, and passing of parameters is the same as the rules described in **5.1 Procedure Division Clause**.

Code Sample

Note: compile the two programs below with -b, e.g.

```
cobc -b entry-test.cbl entry-1.cbl

cobcrun entry-test
```

```
      *
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  ENTRY-TEST.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 FLD-1    PIC X(10) VALUE "CALIFORNIA".
       77 DUMMY    PIC X.
       PROCEDURE DIVISION.
       MAIN.
           DISPLAY "ENTRY-TEST BEGUN!" LINE 5 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.
           CALL "DISP-STATE" USING FLD-1.
           DISPLAY "ENTRY-TEST FINISHED!" LINE 15 COL 10.
           ACCEPT DUMMY LINE 15 COL 30.
           STOP RUN.
      *
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  ENTRY-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 DUMMY   PIC X.
       LINKAGE SECTION.
       01 FIELD-1     PIC X(10).
       PROCEDURE DIVISION.
       MAIN.
           DISPLAY "SUBPGM-1 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.
           STOP RUN.

       ENTRY "DISP-STATE" USING FIELD-1.
```

```
           DISPLAY FIELD-1 LINE 10 COL 10.
           DISPLAY "ENTRY-1 FINISHED!" LINE 11 COL 10.
           ACCEPT DUMMY LINE 11 COL 30.
           EXIT PROGRAM.
```

5.3.17 EVALUATE statement

The EVALUATE Statement allows for different conditions to evaluated.

General Format

```
EVALUATE {evaluate-subject-1} [ ALSO {evaluate-subject-2} ] ...
         {TRUE  }                  {TRUE  }
         {FALSE }                  {FALSE }
   { { WHEN when-condition-1 [ ALSO when-condition-2 ] ... } ...
         statement-1 } ...
     [ WHEN OTHER
         statement-2 ]
   [ END-EVALUATE ]
```

when-condition-n can be any of the following:

```
      {  ANY }
      { (NOT] TRUE }
      { [NOT] FALSE }
      { [NOT] NUMERIC }
      { [NOT] ALPHABETIC }
      { [NOT] ALPHABETIC_LOWER }
      { [NOT] ALPHABETIC_UPPER }
      { [NOT] POSITIVE }
      { [NOT] NEGATIVE }
      { [NOT] = cond-obj }
      { [NOT] EQUAL TO cond-obj }
      { [NOT] EQUALS cond-obj }
      { [NOT] = obj-value-1 {THRU   } obj-value-2 }
                            {THROUGH}
      { [NOT] >cond-obj }
      { [NOT] GREATER THAN cond-obj }
      { [NOT] <cond-obj }
      { [NOT] LESS THAN cond-obj }
      { [NOT] >= cond-obj }
      { [NOT] GE cond-obj }
      { [NOT] GREATER THAN OR EQUAL TO cond-obj }
      { [NOT] <= cond-obj }
      { IS [NOT] LESS THAN OR EQUAL TO cond-obj }
      { < > cond-obj }
      { < > cond-obj }
```

Syntax

1. evaluate-subject-n is a literal, data element, arithmetic expression or conditional expression.
2. condition-n is a condition name described in a level 88 statement.
3. statement-n is an imperative statement.

General Rules

1. evaluate-subject-n can be a literal, data element, arithmetic expression or conditional expression.
2. evaluate-subject-1 follows the EVALUATE statement, and serves as the basis of comparison for all subsequent WHEN condition statements.
3. The EVALUATE subject can combine multiple evaluate-subject-n by means of the ALSO phrase.
4. When the ALSO phrase is used, in an EVALUATE statement, WHEN statements must contain an equal number of ALSO phrases, which all must test TRUE against the ALSO statements in the same ordinal position in the EVALUATE statement in order for the WHEN statement to test TRUE.
5. When-condition is evaluated against evaluate-subject. If when-condition tests TRUE, then statement-1 is executed.
6. If when-condition does not test TRUE when matched with evaluate-subject, then subsequent WHEN statements are evaluated.
7. If no WHEN statements tests TRUE, and a WHEN OTHER statement exists, then the WHEN OTHER condition is TRUE, and statement-2 executes.
8. If a WHEN clause is empty, it tests false, and the next WHEN statement is evaluated.
9. If no WHEN OTHER statement exists, then the EVALUATE statement ends, and control passes to the next statement in the program.
10. THRU and THROUGH are synonyms.
11. The THRU phrase operates with the same General Rules as the THRU phrase applied to the VALUE clause.
12. The NOT phrase causes all values to test TRUE that are not in the range of values following the NOT phrase.
13. The ANY phrase tests TRUE for all comparisons.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  EVALUATE-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 RANDOM-NUMBER     PIC9(9) VALUE 95.
       77 DUMMY             PIC X.
       PROCEDURE DIVISION.
       MAIN.
           EVALUATE RANDOM-NUMBER
             WHEN < 100
               DISPLAY "A SMALL RANDOM NUMBER: " LINE 5 COL 10
```

```
            DISPLAY RANDOM-NUMBER LINE 5 COL 32
          WHEN OTHER
            DISPLAY "A LARGE RANDOM NUMBER" LINE 5 COL 10
            DISPLAY RANDOM-NUMBER LINE 5 COL 32
        END-EVALUATE.

        DISPLAY "EVALUATE-1 FINISHED!" LINE 10 COL 10.
        ACCEPT DUMMY LINE 10 COL 30.

        STOP RUN.
```

5.3.18 EXHIBIT NAMED Statement

The EXHIBIT NAMED statement provides a way to extract the value of a named variable and write it to an error file. Note- Set the runtime environment variable COB_ERROR_FILE, when running a program with the EXHIBIT Statement.

General Format

```
EXHIBIT [NAMED] [element-1]
```

Syntax

1. element-1 is a numeric or alphanumeric data item.

General Rules

1. When the EXHIBIT NAMED statement is applied to a variable, output is written to the Console in the following format:
   [element-1] = [value] . In the example below:
   ELEMENT-1 = 1

Code Sample

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  EXHIBIT-1.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 ELEMENT-1  PIC 9 VALUE 1.
77 DUMMY PIC X.
PROCEDURE DIVISION.
MAIN.
    DISPLAY "EXHIBIT NAMED EXAMPLE" LINE 2 COL 2.
    EXHIBIT NAMED ELEMENT-1.
    ACCEPT DUMMY LINE 10 COL 10.
```

```
           STOP RUN.
```

5.3.19 EXIT Statement

The EXIT statement provides an end point within a PERFORM statement, PARAGRAPH, SECTION, or PROGRAM. When EXITing a PROGRAM, control is returned to the calling program.

Format 1

```
EXIT
```

Format 2

```
EXIT PROGRAM {RETURNING} [identifier-1]
```

Format 3

```
EXIT {PARAGRAPH}
     {SECTION  }
```

Format 4

```
EXIT PERFORM [ CYCLE ]
```

Syntax

1. Identifier-n is a numeric or alphanumeric literal, or data item.

General Rules

1. The Format 1 EXIT statement must be the only statement in the paragraph.
2. The Format 1 EXIT statement performs no operation. Control passes to the next line of executable code in the currently running program.
3. If the currently running program is a CALL'ed subroutine, the Format 2 EXIT PROGRAM statement terminates the execution of the subroutine, and returns control to the CALL'ing program on the next instruction after the CALL statement.
4. In a Format 2 EXIT PROGRAM [RETURNING] identifier-1 statement, the RETURNING phrase is optional.
5. The Format 2 EXIT PROGRAM [RETURNING] identifier-1 statement MOVEs identifier-1 to the special return-code register, and then executes an EXIT PROGRAM statement.

6. If the currently running program is not a CALL'ed subroutine, the Format 2 EXIT PROGRAM statement performs no operation. Control passes to the next line of executable code in the currently running program.
7. When the Format 2 EXIT PROGRAM statement also contains a RETURNING clause, the value of numeric-data-1 is assigned to the RETURN-CODE register, where it can be accessed in the CALL'ing program.
8. The Format 3 EXIT PARAGRAPH/EXIT SECTION statement causes control to be transferred to the next executable statement after the end of the current PARAGRAPH / SECTION.
9. The Format 4 EXIT PERFORM [CYCLE] statement causes control to be transferred to the point just before the matching END-PERFORM. In PERFORM loops with tests, this will allow for the next logical test to take place.

Code Sample

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  EXIT-1.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77 RTN-CODE  PIC 9 VALUE 1.
 77 DUMMY PIC X.
 PROCEDURE DIVISION.
 MAIN.
*FORMAT 1
*    EXIT
*FORMAT 2
*    EXIT PROGRAM [ RETURNING NUMERIC-ITEM ]
*FORMAT 3
*    EXIT {PARAGRAPH}
*         {SECTION  }
*FORMAT 4
*    EXIT PERFORM [ CYCLE ]

 DISPLAY "EXIT-1 FINISHED!" LINE 11 COL 10.
 ACCEPT DUMMY LINE 11 COL 30.
 STOP RUN.
 EXIT-THE-PROGRAM.
     EXIT.
 EXIT-PGM-RETURNING.
     EXIT PROGRAM RETURNING RTN-CODE.
 EXIT-PARA.
     EXIT PARAGRAPH.
     EXIT SECTION.
 EXIT-SECT.
     EXIT SECTION.
 EXIT-PFRM.
     PERFORM FOREVER
       IF RTN-CODE = 1
         EXIT PERFORM CYCLE
       END-IF
     END-PERFORM.
```

5.3.20 FREE Statement

The FREE statement releases memory allocated using the ALLOCATE statement.

General Format

```
FREE { data-ptr-1 } …
```

Syntax

1. data-ptr-n is a data item declared with USAGE POINTER.

General Rules

1. The FREE statement releases the memory referenced by data-ptr-1.
2. Any BASED data item using data-ptr-1 as a memory reference no longer has a memory reference pointer.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  FREE-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 DATA-PTR-1           USAGE POINTER BASED.
       77 DATA-PTR-2           USAGE POINTER.
       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
      *FREEDATA-PTR
      *
            ALLOCATE DATA-PTR-1 INITIALIZED.
            ALLOCATE 100 CHARACTERS INITIALIZED RETURNING DATA-PTR-2.

            DISPLAY "ALLOCATE SUCCESSFUL!" LINE 10 COL 10.

            FREE DATA-PTR-1, DATA-PTR-2.

            DISPLAY "FREE-1 FINISHED!" LINE 11 COL 10.
            ACCEPT DUMMY LINE 11 COL 30.
            STOP RUN.
```

5.3.21 GO TO Statement

The GO TO statement transfers controls to a named paragraph or section in the Procedure Division.

Format 1

GO TO procedure-name

Format 2

GO TO { proc-1 } ... DEPENDING ON numeric-data-1

Syntax

1. proc-n is the name of a procedure or section in the program.
2. numeric-data-n is a numeric data item.

General Rules

1. The Format 1 GOTO statement transfers control to the named paragraph or section. If a section is named, control is transferred to the first paragraph in the section.
2. In a Format 2 GOTO {list of paragraphs} DEPENDING statement, a list of paragraphs is followed by a numeric data tem. The ordinal position of the paragraph in the list must correspond with the value of numeric-data-1 in order for it to be selected as the target of the GOTO statement. In the Code Sample below, the GOTO statement is followed by two procedure names, but since the GOTO-DEP-VAR is set to 1, the GOTO statement will transfer control to the first one.
3. If the value of numeric-data-1 is less than 1 or greater than the number of listed GOTO targets, the operation is not executed, and control passes to the next statement.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  GOTO-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 GOTO-FLAG PIC 9 VALUE 1.       77 GOTO-DEP-VAR PIC 9 VALUE 1.
       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
      *    GO TO PROCEDURE-NAME
            IF GOTO-FLAG = 1
              GOTO GOTO-1-EXIT
            END-IF.

      *    GO TO { PROCEDURE-NAME } ... DEPENDING ON IDENTIFIER
            GOTO DEP-1-EXIT, DEP-2-EXIT DEPENDING ON GOTO-DEP-VAR.
       77 GOTO-DEP-VAR PIC 9 VALUE 1.
       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
      *    GO TO PROCEDURE-NAME
            IF GOTO-FLAG = 1
              GOTO GOTO-1-EXIT
            END-IF.

      *    GO TO { PROCEDURE-NAME } ... DEPENDING ON IDENTIFIER
            GOTO DEP-1-EXIT, DEP-2-EXIT DEPENDING ON GOTO-DEP-VAR.

       GOTO-1-EXIT.
            DISPLAY "GOTO-1 EXIT!" LINE 10 COL 10.
            ACCEPT DUMMY LINE 10 COL 30.
            STOP RUN.
       DEP-1-EXIT.
```

```
           DISPLAY "DEP-1 EXIT!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.
           STOP RUN.
       DEP-2-EXIT.
           DISPLAY "DEP-2 EXIT!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.
           STOP RUN.
```

5.3.22 GOBACK Statement

The GOBACK statement ends a program. If the program is a CALL'ed program, control is returned to the CALLing program. Otherwise, the program terminates.

General Format

```
GOBACK [RETURNING] [identifier-1].
```

Syntax

Identifier-n is a numeric or alphanumeric literal or data item.

General Rules

1. If the currently running program is a CALL'ed subroutine, the GOBACK statement terminates the execution of the subroutine, and returns control to the CALL'ing program on the next instruction after the CALL statement.
2. If the currently running program is not a CALL'ed subroutine, the GOBACK statement performs the STOP RUN operation, terminating the current runtime session.
3. The statement GOBACK RETURNING identifier-1 MOVEs identifier-1 to the special return-code register, and then performs the GOBACK statement.
4. In the statement GOBACK RETURNING identifier-1, the RETURNING phrase is optional.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  GOBACK-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 DUMMY              PIC X.
       PROCEDURE DIVISION.
       MAIN.
           DISPLAY "GOBACK-1 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.
           GOBACK.
```

5.3.23 IF statement

The IF statement is a conditional statement that evaluates the truth of a condition, and provides alternative branches for the runtime to follow depending on whether the condition is true or false.

General Format

```
IF condition-1  [THEN] { statement-list-1 } …
     [ ELSE {statement-list-2 } …  ]
     [ END-IF ]
```

Syntax

1. condition-n is a condition name described in a level 88 statement.
2. statement-list-n consists of one or more imperative or conditional statements.

General Rules

1. If condition-1 is TRUE, statement-list-1 executes, and control is then passed to the next executable statement.
2. If condition-1 is FALSE, and if an ELSE clause exists, then statement-list-2 executes. If no ELSE clause exists, then control is transferred to the end of the IF statement.
3. An IF statement can be terminated by :
    a. A period ".".
    b. An END-IF statement at the same nesting level.
    c. An ELSE phrase in an IF statement at a higher nesting level.
4. statement-list-2 is optional. If there is no statement-list following an ELSE phrase, the end of the scope of the ELSE statement is reached, and control passes to a higher nesting level, or to the statement following the IF statement.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  IF-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 NUMERIC-1     PIC9(5)V9(5) VALUE 10.
       77 NUMERIC-2     PIC9(5)V9(5) VALUE 20.
       77 DUMMY  PIC X.
       PROCEDURE DIVISION.
       MAIN.
           IF NUMERIC-1 > NUMERIC-2
             THEN
               DISPLAY "YOU WIN!" LINE 10 COL 10
             ELSE
               DISPLAY "YOU LOSE!" LINE 10 COL 10
               DISPLAY "CHANGE VALUES AND TRY AGAIN." LINE 11 COL 10
```

```
            END-IF.

            ACCEPT DUMMY LINE 11 COL 40.
            STOP RUN.
```

5.3.24 INITIALIZE statement

The INITIALIZE statement sets data items or categories of data items to prescribed values.

General Format

```
    INITIALIZE { data-1 } … [ WITH FILLER ]
           [REPLACING  {initialize_category} DATA BY identifier-1 }... ]
```

where **initialize_category** is one of the following:

```
ALPHABETIC
ALPHANUMERIC
NUMERIC
ALPHANUMERIC-EDITED
NUMERIC-EDITED
NATIONAL
NATIONAL-EDITED
```

Syntax

1. data-n is a data item.
2. identifier-n is a data element, literal, or data returned from a function call.

General Rules

1. The INITIALIZE statement can be used to target data elements or classes of data elements. The initialize-category Classes of data elements that are recognized by the INITIALIZE statement are:
    a. ALPHABETIC
    b. ALPHANUMERIC
    c. NUMERIC
    d. ALPHANUMERIC-EDITED
    e. NUMERIC-EDITED
    f. NATIONAL
    g. NATIONAL-EDITED

For more information on Picture Catagories, see 4.8.11 Picture Data Categories.

2. The default behavior of the INITIALIZE verb is to initialize Alphabetic, Alphanumeric, Alphanumeric-Edited, National, and National-Edited data items to SPACES, and Numeric and Numeric-Edited data items to ZERO.
3. If a group item is initialized which contains a mixture of alphanumeric, numeric, and

national data types, each data type will be initialized according to the default behaviours of COBOL-IT.

4. If the REPLACING clause is used, the implied MOVE is evaluated by the compiler to ensure that it is valid.
5. The following data items should not be targets of the INITIALIZE verb:
   a. Data items containing an OCCURS DEPENDING ON clause.
   b. FILLER data items, unless the WITH FILLER clause is present.
   c. Data items with a REDEFINES clause.
   d. Data items described as USAGE INDEX
6. The following compiler configuration file entries affect the behavior of the INITIALIZE statement (See Compiler and Runtime Reference Manual for complete details):
   initialize-to-value:[yes/no]
   initialize-fd:[yes/no]
   initialize-filler:[yes/no]
   initialize-pointer:[yes/no]

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  INITIALIZE-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 DUMMY              PIC X.
       PROCEDURE DIVISION.
       MAIN.

      *INITIALIZE { TARGET } … [ WITH FILLER ]
      *[REPLACING  {INITIALIZE-CATEGORY} DATA BY VALUE }... ]
      *
      *WHERE INITIALIZE-CATEGORY IS ONE OF THE FOLLOWING:
      *
      *ALPHABETIC
      *ALPHANUMERIC
      *NUMERIC
      *ALPHANUMERIC-EDITED
      *NUMERIC-EDITED
      *NATIONAL
      *NATIONAL-EDITED

           INITIALIZE DUMMY.

           INITIALIZE DUMMY WITH FILLER.

           INITIALIZE DUMMY
           REPLACING ALPHABETIC DATA BY "Z"
                     ALPHANUMERIC DATA BY "Y"
                     NUMERIC DATA BY 0
                     ALPHANUMERIC-EDITED DATA BY "X"
                     NUMERIC-EDITED DATA BY "W"
                     NATIONAL DATA BY "V"
                     NATIONAL-EDITED DATA BY "U".

           DISPLAY "INITIALIZE-1 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 35.
           STOP RUN.
```

5.3.25 INSPECT Statement

The INSPECT statement provides TALLYING, REPLACING, and CONVERTING functions for counting and modifying elements in character strings within a program.

Format 1 :

```
inspect identifier-1
[ tallying tallying-phrase   ]
[ replacing replacing-phrase ]
[ after-before-phrase        ]
```

Format 2 :

```
inspect identifier-1
[converting converting-phrase    ]
[after-before-phrase  ]
```

Syntax

1. identifier-n is a data element, literal, or data returned from a function call.

Tallying phrase format:

```
tallying numeric-data-1 for  [ CHARACTERS ]
                             [ { ALL       }
                               { LEADING   }  identifier-2 ]
                               { FIRST     }
                               { TRAILING  }
```

Syntax

1. numeric-data-n is a numeric data item
2. identifier-n is a data element, literal, or data returned from a function call.

General Rules

1. The TALLYING clause is designed to count the number of CHARACTERS, or the iterations of a character string found in identifier-1, or in a subset of the characters in identifier-1 as

defined by the after-before clause.

2. The CHARACTERS phrase causes a simple count of the number of characters in the named identifier to be tallied, and stored in numeric-data-1.
3. The ALL phrase causes a count of the number of iterations of identifier-2 in the named identifier to be tallied, and stored in numeric-data-1.
4. The LEADING phrase causes a count of the number of contiguous iterations of identifier-2, starting at the left-most position in the named identifier.
5. The FIRST phrase causes numeric-data-1 to be incremented if/when the first iteration of identifier-2 is located in the named identifier.
6. The TRAILING phrase causes a count of the number of contiguous iterations of identifier-2, starting at the right-most position in the named identifier, and parsing from right to left.

Replacing phrase format:

```
replacing   [ CHARACTERS ]
            [ { ALL      }
              { LEADING  }  identifier-3 ] BY identifier-4
              { FIRST    }
              { TRAILING }
```

Syntax

1. identifier-n is a data element, literal, or data returned from a function call.

General Rules

1. The REPLACING clause is designed to identify a string of characters in identifier-1 or in a subset of the characters in identifier-1 as defined by the after-before clause, and replace them with an alternative string of characters of the same size.
2. The CHARACTERS phrase causes every character in identifier-1 to be replaced by identifier-4. When using the CHARACTERS phrase, identifier-3 is an alphanumeric data item that is a single character.
3. The ALL phrase causes all instances of identifier-3 inside identifier-1 to be replaced by identifier-4.
4. The LEADING phrase causes all contiguous iterations of identifier-3 inside identifier-1, starting at the left-most position in identifier-1 to be replaced by identifier-4.
5. The FIRST phrase causes the first iteration of identifier-3 to be replaced by identifier-4.
6. The TRAILING phrase causes all contiguous iterations of identifier-3 inside identifier-1, starting at the right-most position in the named identifier, and parsing from right to left, to be replaced with identifier-4.
7. Note that when using the ALL, LEADING, FIRST, and TRAILING phrases, identifier-3 and identifier-4 must be strings of the same length.

after-before phrase format:

```
[ { BEFORE } INITIAL identifier-5 ]
```

```
{ AFTER  }
```

Syntax

1. identifier-n is a data element, literal, or data returned from a function call.

General Rules

1. The optional AFTER-BEFORE phrase specifies the end-point or beginning-point in the named identifier that will be parsed in the INSPECT statement.
2. The BEFORE phrase causes the INSPECT/TALLYING/REPLACING statement to locate a parsing end-point in the named identifier. Parsing begins at the first byte of the named identifier, and ends when the string named in identifier-5 is located.
3. The AFTER phrase causes the INSPECT/TALLYING/REPLACING statement to locate a parsing beginning-point in the named identifier. The named identifier is scanned from right to left. If an identifier-5 is located, it marks the parsing beginning-point, and parsing continues to the end of the identifier.

CONVERTING phrase format:

```
[converting identifier-2 TO identifier-3]
[ after-before-phrase                   ]
```

Syntax

1. identifier-n is a data element, literal, or data returned from a function call.

General Rules

1. The CONVERTING phrase dictates a conversion of characters within identifier-1 according to rules established by the ordinal positions of characters in identifier-2 and identifier-3. To clarify with an example, the statement :
2. INSPECT identifier-1 CONVERTING “AB” TO “CD” causes every instance of “A” in identifier-1 (the first character in identifier-2), to be converted to “C” (the first character in identifier-3), and causes every instance of “B” in identifier-1 (the second character in identifier-2) to be replaced by “D” (the second character in identifier-3.
3. To further clarify, INSPECT CONVERTING is commonly used to convert lower-case characters to upper-case, as follows:
   INSPECT identifier-1 CONVERTING “abcdefghijklmnopqrstuvwxyz” to “ABCDEFGHIJKLMNOPQRSTUVWXYZ”.
   The effect of this statement is to convert every lower case letter in identifier-1 to its corresponding upper-case letter.

after-before phrase format:

```
[ { BEFORE } INITIAL identifier-5 ]
  { AFTER  }
```

Syntax

1. identifier-n is a data element, literal, or data returned from a function call.

General Rules

1. The optional AFTER-BEFORE phrase specifies the end-point or beginning-point in the named identifier that will be parsed in the INSPECT statement.
2. The BEFORE phrase causes the INSPECT/CONVERTING statement to locate a parsing end-point in the named identifier. Parsing begins at the first byte of the named identifier, and ends when the string named in identifier-6 is located.
3. The AFTER phrase cases the INSPECT/CONVERTING statement to locate a parsing beginning-point in the named identifier. The named identifier is scanned from right to left. If an identifier-6 is located, it marks the parsing beginning-point, and parsing continues to the end of the identifier.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  INSPECT-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 TALLY-CTR1  PIC 99 VALUE 0.
       77 STRING-1  PIC X(10) VALUE "COBOL-IT".
       77 STRING-2  PIC X(20) VALUE "OPEN SOURCE COBOL".
       77 STRING-3  PIC X(42)
          VALUE "A QUICK BROWN FOX JUMPED OVER THE LAZY DOG".
       77 DUMMY     PIC X.
       PROCEDURE DIVISION.
       MAIN.
           INSPECT STRING-1 TALLYING TALLY-CTR1 FOR ALL "O".
           DISPLAY "THERE ARE " TALLY-CTR1 " O'S IN " STRING-1
             LINE 5 COL 10.

           INSPECT STRING-2 REPLACING ALL "E" BY "-".
           DISPLAY STRING-2 LINE 6 COL 10.

           INITIALIZE TALLY-CTR1.
           INSPECT STRING-2 TALLYING TALLY-CTR1 FOR ALL "O"
             REPLACING ALL "S" BY "+".
           DISPLAY STRING-2 LINE 7 COL 10.
           DISPLAY "THERE ARE " TALLY-CTR1 " O'S IN " STRING-2
             LINE 8 COL 10.

           DISPLAY STRING-3 LINE 10 COL 10.
           DISPLAY "CONVERTING TO UPPER-CASE...." LINE 11 COL 10.
           INSPECT STRING-3 CONVERTING
             "ABCDEFGHIJKLMNOPQRSTUVWXYZ" TO
             "abcdefghijklmnopqrstuvwxyz".
```

```
           DISPLAY STRING-3 LINE 12 COL 10.

           DISPLAY "INSPECT-1 FINISHED!" LINE 15 COL 10.
           ACCEPT DUMMY LINE 15 COL 30.
           STOP RUN.
```

5.3.26 MERGE Statement

The MERGE statement merges multiple files with identical key layouts into a single file.

General Format

```
MERGE sort-file-1
{ ON  {ASCENDING} KEY { sort-key-1 … } …
      {DESCENDING}
[COLLATING SEQUENCE {IS alphabet-name-1}
[WITH DUPLICATES IN ORDER]
USING file-2 { file-3}...
GIVING {file-4} …
{OUTPUT PROCEDURE IS proc-1 [{THROUGH} proc-2]}
                             {THRU    }
```

Syntax

1. sort-file-n is a file described in the File Section with an SD.
2. sort-key-n is a data element that has been named as a key for the SORT.
3. alphabet-name is a user-defined word.
4. file-n is a file described in the File Section with an FD.
5. proc-n is the name of a procedure or section in the program.

General Rules

Overview of the MERGE Statement.

The MERGE Statement is provided with a number of input files (USING files), all sorted on identical key layouts. The MERGE statement OPENs all of the USING files, READs the first record of each USING file, compares the records on the KEY, selects a record according to the KEY information, and WRITEs it to the temporary merge-file. When all of the records have been merged into the merge-file, they are transferred into the output file(s) either through the GIVING clause or the RETURN statement in an OUTPUT PROCEDURE.

When all of the records have been merged into the merge-file, there are two ways to return records from the merge-file into an output file. The simplest way, from a syntactic standpoint, is with the USING/GIVING syntax. This requires a statement, such as :

```
           MERGE ALLSTAR-MERGE
```

```
      *MAJOR SORTKEY
            ON ASCENDING KEY MEMBER-LAST-NAME
      *MINOR SORTKEY
           ON DESCENDING KEY MEMBER-TEAM
      *DUPLICATES SORTED IN THE ORDER ACQUIRED
           WITH DUPLICATES IN ORDER
           USING  BASEBALL-FILE, HOOP-FILE
           GIVING ALLSTAR-FILE.
```

In this case, the MERGE statement uses the ALLSTAR-MERGE file as the temporary sort file, into which the records are written before they are transferred to the GIVING file.

A slightly more complex alternative, which provides the user with more flexibility involves using the OUTPUT PROCEDURE syntax. This requires a statement, such as:

```
       77 OUTPUT-MERGE-AT-END  PIC X.
         88 EOF-MERGE-FILE VALUE "Y".

       .....

           MERGE ALLSTAR-MERGE
      *MAJOR SORTKEY
            ON ASCENDING KEY MEMBER-LAST-NAME
      *MINOR SORTKEY
           ON DESCENDING KEY MEMBER-TEAM
      *DUPLICATES SORTED IN THE ORDER ACQUIRED
           WITH DUPLICATES IN ORDER
           USING  BASEBALL-FILE, HOOP-FILE
           OUTPUT PROCEDURE OUTPUT-PROC.

       .....

       OUTPUT-PROC SECTION.
       OUTPUT-PROCESS.
           OPEN OUTPUT ALLSTAR-FILE.

           PERFORM UNTIL EOF-MERGE-FILE
             RETURN ALLSTAR-MERGE
               AT END MOVE "Y" TO OUTPUT-MERGE-AT-END
               NOT AT END
                 WRITE ALLSTAR-INFO FROM MERGE-DATA
             END-RETURN
           END-PERFORM.
```

In this case, the MERGE statement uses the ALLSTAR-MERGE file as the temporary sort file, uses the OUTPUT PROCEDURE as an area where processing can be done, and controls the writing to the output file through the use of the RETURN statement.

Note that while the input files, and output file are described in the FILE SECTION of the program with FD's, the ALLSTAR-MERGE file must be described in the FILE SECTION of the program with an SD.

General Rules

1. The MERGE statement performs the equivalent of OPEN INPUT for all of the USING files.

After the MERGE is complete, the MERGE statement CLOSEs the USING and GIVING files.

2. Sort-file-1 must be described with an SD in the FILE SECTION.
3. The USING and GIVING files must be described with ORGANIZATION LINE SEQUENTIAL, or ORGANIZATION BINARY SEQENTIAL, and must be described with an FD in the FILE SECTION.
4. The order in which the KEY clauses are listed indicates the order in which the sort keys are applied. The first KEY listed is the primary key, the second key listed is the second key, etc...
5. The ASCENDING KEY clause causes values to sort from lowest to highest.
6. The DESCENDING KEY clause causes values to sort from highest to lowest.
7. If an ASCENDING/ DESCENDING clause has been used, it will be applied to subsequent KEY clauses until an alternative DESCENDING/ASCENDING clause is used.
8. The COLLATING SEQUENCE clause names the alphabet which is used for purposes of ordering KEY data in the MERGE. Usage of the COLLATING SEQUENCE clause in a MERGE statement overrides the naming of a PROGRAM COLLATING SEQUENCE in the object-computer paragraph.
9. The MERGE statement writes duplicates in the order in which they are encountered. That is, if there are two USING files, and the current records in each file have duplicate keys, then the order of the listing of the USING files determines which record is written first.
10. Files listed in the USING clause may not be OPEN when the MERGE statement executes. The MERGE statement OPENs the files.
11. All of the input files must be sorted on identical keys, as described by the KEY clause of the MERGE statement.
12. Files listed in the GIVING clause may not be OPEN when the MERGE statement executes. The MERGE statement OPENs the files.
13. The OUTPUT PROCEDURE is executed after all of the files have been merged into the temporary merge-file.
14. Inside the OUTPUT PROCEDURE, the RETURN statement is used to transfer records from the merge-file to the output file.
15. When the OUTPUT PROCEDURE is finished, the MERGE statement closes all files.
16. The MERGE statement updates file status variable that is defined for merge-file after performing OPEN, READ, WRITE, and CLOSE operations. File Status errors may be detected in DECLARATIVES.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. MERGE-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
      *MERGE INPUT FILE
           SELECT BASEBALL-FILE ASSIGN TO "BASEBALLSTARS.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS BASEBALL-STAT.

      *MERGE INPUT
           SELECT HOOP-FILE ASSIGN TO "HOOPSTARS.TXT"
```

```
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS HOOP-STAT.

       *MERGE OUTPUT FILE
           SELECT ALLSTAR-FILE ASSIGN TO "ALLSTARS.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS ALLSTARS-STAT.

       *MERGE FILE (SD)
           SELECT ALLSTAR-MERGE ASSIGN TO "MERGE-WORK".

        DATA DIVISION.
        FILE SECTION.
        FD BASEBALL-FILE.
        01 BASEBALL-INFO PIC X(40).

        FD HOOP-FILE.
        01 HOOP-INFO PIC X(40).

        FD ALLSTAR-FILE.
        01 ALLSTAR-INFO PIC X(40).

        SD ALLSTAR-MERGE.
        01 MERGE-DATA.
          05 MEMBER-FIRST-NAME PIC X(10).
          05 MEMBER-LAST-NAME  PIC X(15).
          05 MEMBER-TEAM PIC X(15).

        WORKING-STORAGE SECTION.
        77 BASEBALL-STAT PIC XX.
        77 HOOP-STAT PIC XX.
        77 ALLSTARS-STAT PIC XX.
        77 DUMMY         PIC X.

        PROCEDURE DIVISION.
        STAR-MERGE.
           MERGE ALLSTAR-MERGE
       *MAJOR SORTKEY
            ON ASCENDING KEY MEMBER-LAST-NAME
       *MINOR SORTKEY
           ON DESCENDING KEY MEMBER-TEAM
       *DUPLICATES SORTED IN THE ORDER ACQUIRED
           WITH DUPLICATES IN ORDER
           USING  BASEBALL-FILE, HOOP-FILE
           GIVING ALLSTAR-FILE.

           DISPLAY "MERGE-1 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.
           STOP RUN.
```

→baseballstars.txt

```
BABE      RUTH           YANKEES
LOU       GEHRIG         YANKEES
JOE       DIMAGGIO       YANKEES
ENOS      SLAUGHTER      CARDINALS
DOMINIC   DIMAGGIO       RED SOX
JOHNNY    BENCH          REDS
```

```
BARRY     BONDS          GIANTS
HANK      AARON          BRAVES
```

→hoopstars.txt

```
LARRY     BIRD           CELTICS
MAGIC     JOHNSON        LAKERS
BILL      RUSSELL        CELTICS
JULIUS    ERVING         SIXERS
SPUD      WEBB           HAWKS
MOSES     MALONE         SIXERS
KC        JONES          CELTICS
MICHAEL   JORDAN         BULLS
```

5.3.27 MOVE Statement

The MOVE statement transfers data stored in a source location to one or more named target locations.

Format 1

```
MOVE identifier-1 TO {data-1} …
```

Format 2

```
MOVE {CORRESPONDING} src-group-item-1 TO target-group-item-1
     {CORR          }
```

Syntax

1. identifier-n is a data element, literal, or data returned from a function call.
2. data-n is a data item.
3. Src-group-item-n and target-group-item-n are group data items containing one or more elementary data items.

General Rules

1. A FORMAT 1 MOVE statement may name multiple target data fields.
2. A FORMAT 1 MOVE from a group item to another group item is treated as a MOVE of an alphanumeric data item to an alphanumeric data item. No check of the validity of the moves at the elementary level is made.
3. MOVEs to and from USAGE DISPLAY data items perform data conversion using the CODEPAGE module.
4. The Format 1 MOVE statement allows for the source of the MOVE statement to be MOVEd to multiple targets. Two sytaxes are supported for a MOVE to multiple targets. Consider the cases where a variable a is being MOVEd to three target variables b, c, and d:
    a. MOVE a TO b,c,d.
    b. MOVE a TO b TO c TO d.

The two statements above are functionally equivalent. That is, in each case, the field a is the source field for the MOVE statement, and the value of a is MOVEd separately to each of the three target variables b, c, and d.

5. A FORMAT 2 MOVE statement copies the contents of elements in src-group-item to elements in target-group-item which are identified as CORRESPONDING data elements.
    a. For details on how data items are identified as CORRESPONDING, see 5.3.1 Common General Rules / Rules for identifying CORRESPONDING data elements.
    b. A MOVE CORRESPONDING statement first identifies all CORRESPONDING data elements in the src-group-item and target-group-item, and then performs a series of MOVEs from the identified data elements in the src-group-item to their CORRESPONDING data elements in the target-group-item.
    c. CORRESPONDING and CORR are synonyms.
6. Rules for justification, space filling, decimal point alignment, and zero filling in the target-data-element are described in 4.8.12 Rules for Alignment of Data.
7. When evaluating the validity of a MOVE statement, the compiler assigns data items/literals to PICTURE DATA CATEGORIES, and then applies certain pre-set rules.
8. In certain cases, the compiler will report an “Error: Invalid MOVE statement”, and abort the compilation:
    a. ALPHABETIC data items cannot be MOVEd to NUMERIC or NUMERIC EDITED data items.
    b. NUMERIC/NUMERIC EDITED data items be cannot be MOVEd to ALPHABETIC data items.
    c. ALPHANUMERIC data items cannot be MOVEd to NUMERIC or NUMERIC EDITED data items.
    d. ALPHANUMERIC EDITED data items cannot be MOVEd to NUMERIC or NUMERIC EDITED data items.
    e. NUMERIC non-integer data items cannot be MOVEd to ALPHANUMERIC or ALPHANUMERIC EDITED data items.
    f. NUMERIC EDITED data items cannot be MOVE’d to ALPHANUMERIC data items.
    g. For details on the different PICTURE DATA CATEGORIES, see 4.8.11 Picture Data Categories. Literals are assigned DATA CATEGORIES according to the following rules:
9. Numeric literals are numeric.
10. All other literals and figurative contants are alphanumeric, with the following exceptions:
    a. ZERO is numeric
    b. SPACE is alphabetic

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  MOVE-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 NUMERIC-1     PIC9(5)V9(5) VALUE10.
       77 NUMERIC-2     PIC9(5)V9(5) VALUE20.
       01 GROUP-1.
         03 FLD-1       PIC 99 VALUE 10.
```

```
          03 FLD-2        PIC 99 VALUE 20.
       01 GROUP-2.
          03 FLD-1        PIC 99.
          03 FLD-2        PIC 99.
       77 DUMMY  PIC X.
       PROCEDURE DIVISION.
       MAIN.
           MOVE NUMERIC-1 TO NUMERIC-2.
           MOVE CORRESPONDING GROUP-1 TO GROUP-2.
           ACCEPT DUMMY LINE 15 COL 30.
           STOP RUN.
```

5.3.28 MULTIPLY Statement

The MULTIPLY statement performs arithmetic multiplication, allowing for the storage of the product in one or more data items.

Format 1

The Format 1 MULTIPLY Statement MULTIPLYs an initial numeric data item BY one or more numeric data items , and moves the result of the MULTIPLY operation into the initial numeric data item. The result may be optionally ROUNDED.

```
MULTIPLY numeric-1 BY { numeric-data-2 [ROUNDED] } ...
     [ ON SIZE ERROR statement-1 ]
     [ NOT ON SIZE ERROR statement-2 ]
     [ END-MULTIPLY ]
```

Format 2

The Format 2 MULTIPLY Statement MULTIPLYs an initial numeric data item BY a second numeric data item , and moves the result of the MULTIPLY operation into the data element(s) following the GIVING clause. The result may be optionally ROUNDED.

```
MULTIPLY numeric-3 BY numeric-data-4
     GIVING { numeric-data-5 [ROUNDED] } ...
     [ ON SIZE ERROR statement-1 ]
     [ NOT ON SIZE ERROR statement-2 ].
     [ END-MULTIPLY ]
```

Syntax

1. numeric-n is a literal or data item that is numeric.
2. numeric-data-n is a numeric data item.
3. statement-n is an imperative statement.

General Rules

1. Format 1, and Format 2 of the MULTIPLY operation allow multiple receiving fields.

The ROUNDED clause is applied when a MULTIPLY operation produces a result that includes more decimal places than are included in the description of the data item given to hold the final result of the arithmetic operation.

2. For rules regarding the ROUNDED clause, see the entry for ROUNDED in 5.3.1 Common General Rules.
3. The SIZE ERROR exception is triggered if the ON SIZE ERROR clause is present and if the receiving field is not large enough to accommodate the result of the MULTIPLY function.
4. For rules regaring the ON SIZE ERROR clause, see the entry for ON SIZE ERROR in 5.3.1 Common General Rules.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  MULTIPLY-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       78 CONST-1           VALUE 10.
       78 CONST-2           VALUE 25.
       77 NUMERIC-1         PIC 99.
       77 NUMERIC-2         PIC 99999.
       77 NUMERIC-3         PIC 99999.
       77 DUMMY             PIC X.
       PROCEDURE DIVISION.
       MAIN.
           MOVE CONST-1 TO NUMERIC-1.
           MOVE CONST-2 TO NUMERIC-2.

           MULTIPLY NUMERIC-1 BY NUMERIC-2 ROUNDED
             ON SIZE ERROR
               DISPLAY "FORMAT 1 SIZE ERROR!" LINE 3 COL 10
             NOT ON SIZE ERROR
               DISPLAY NUMERIC-2 LINE 3 COL 10
           END-MULTIPLY.
      *
           MOVE CONST-1 TO NUMERIC-1.
           MOVE CONST-2 TO NUMERIC-2.

           MULTIPLY NUMERIC-1 BY NUMERIC-2
             GIVING NUMERIC-3 ROUNDED
             ON SIZE ERROR
               DISPLAY "FORMAT 2 SIZE ERROR!" LINE 4 COL 10
           NOT ON SIZE ERROR
             DISPLAY NUMERIC-3 LINE 4COL 10
           END-MULTIPLY.

           DISPLAY "MULTIPLY-1 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.

           STOP RUN.
```

5.3.29 OPEN Statement

The OPEN statement places a file in a state where it may be processed.

General Format

```
OPEN {INPUT } [optional] [share-opt][REVERSED] {file-1 [WITH NO REWIND]...}...
     {OUTPUT}                                  [WITH LOCK     ]
     {I-O   }
     {Extend}
```

where share-opt is

```
      SHARING WITH READ ONLY or
      SHARING WITH NO OTHER
```

Syntax

1. file-n is a file described in the File Section with an FD.

General Rules

1. The OPEN statement has 4 modes, INPUT, OUTPUT, I-O, and Extend.
2. The OPEN statement updates the FILE STATUS variable.
3. An attempt to OPEN a file that is already OPEN fails with a file status of 41.
4. Failures of the OPEN statement may be trapped in DECLARATIVES.
5. A file must have performed a successful OPEN statement before it may execute a successful CLOSE, DELETE, READ, START, or REWRITE, UNLOCK, or WRITE statement.
6. A file that has successfully executed an OPEN INPUT, or OPEN I-O statement positions the file pointer at the first logical record.
7. The default scope of the OPEN state of a file is the current program. If the file is described as GLOBAL, the scope of the OPEN state of the file extends to the entire compilation unit, including nested programs. If the file is described as EXTERNAL, the scope of the OPEN state extends to separately compiled programs that are also described as EXTERNAL.
8. The same file can be OPENed and CLOSEd by different programs in the same run unit, except in cases where the file is operating under restrictive file locking rules, as indicated in the SELECT phrase, or in the OPEN statement. File locking considerations are described below:
9. The SHARING clause in the SELECT statement sets a level of restrictiveness on whether a file that has been OPENed can be OPENed by another file connector. Conditions causing a failed OPEN are marked with the word “Fails” , and conditions leading to a successful OPEN are marked with the word “Succeeds” in the table below:

Sharing / OPEN MODE	Sharing WITH NO OTHER	Sharing WITH READ ONLY		Sharing WITH ALL OTHER	
	EXTEND, I-O, INPUT, OUTPUT	EXTEND, I-O, OUTPUT	INPUT	EXTEND, I-O, OUTPUT	INPUT
WITH NO OTHER	EXTEND, I-O, INPUT , OUTPUT	Fails	Fails	Fails	Fails	Fails
WITH READ ONLY	EXTEND, I-O	Fails	Fails	Fails	Fails	Succeeds
	INPUT	Fails	Fails	Succeeds	Fails	Succeeds
	OUTPUT	Fails	Fails	Fails	Fails	Fails
WITH ALL OTHER	EXTEND, I-O	Fails	Fails	Fails	Succeeds	Succeeds
	INPUT	Fails	Succeeds	Succeeds	Succeeds	Succeeds
	OUTPUT	Fails	Fails	Fails	Fails	Fails

10. A SHARING phrase on an OPEN statement takes precedence over a SHARING phrase in a SELECT statement.
11. The WITH NO REWIND phrase is treated as commentary.
12. The WITH LOCK phrase is treated as commentary.

General Rules OPEN INPUT

1. A file that is OPEN INPUT may successfully execute the START, READ, UNLOCK, and CLOSE statements.
2. A file that is OPEN INPUT may **not** make updates to the file via the DELETE, REWRITE, or WRITE statement.
3. A file that has successfully executed an OPEN INPUT statement positions the file pointer at the first logical record.
4. If the file does not exist when the OPEN INPUT statement is executed, then the OPEN statement fails with a FILE STATUS of 35 unless the SELECT phrase of the file contains the OPTIONAL phrase. In this case, the OPEN is SUCCESSFUL, but the first READ fails with an end-of-file status code.

General Rules OPEN INPUT REVERSED

1. OPEN INPUT REVERSE is supported for RECORD SEQUENTIAL files with fixed length records.
2. After performing an OPEN INPUT REVERSE, the READ statements return records in reverse order, beginning with the last record.

Example- In a record sequential file with records of 10 characters, and containing data as follows:

AAAAAAAAAA
BBBBBBBBBB
CCCCCCCCCC

The phrase :
open input log-file reversed.
read log-file into ws-log-record.

Returns the last record, with values CCCCCCCCCC.

General Rules OPEN OUTPUT

1. A file that is OPEN OUTPUT creates a new file in the current directory or in the file designated by the COB_FILE_PATH environment variable.
2. A file that is OPEN OUTPUT may successfully execute the UNLOCK, WRITE, and CLOSE statements.
3. A file that is OPEN OUTPUT may not perform READ statements, or make updates to the file via the DELETE, or REWRITE statement.
4. A file that has successfully executed an OPEN OUTPUT statement positions the file pointer at beginning of the newly created file.
5. If the file does not exist when the OPEN OUTPUT statement is executed, then the OPEN OUTPUT statement creates the file. Note that if the file does exist when the OPEN OUTPUT statement is executed then the existing file is removed, and a new instance of the file is created.

General Rules OPEN I-O

1. A file that is OPEN I-O may successfully execute the READ, WRITE, REWRITE, START, DELETE, UNLOCK, and CLOSE statements.
2. A file that has successfully executed an OPEN I-O statement positions the file pointer at the first logical record.
3. If the file does not exist when the OPEN I-O statement is executed, then the OPEN statement fails with a FILE STATUS of 35 unless the SELECT phrase of the file contains the OPTIONAL phrase. In this case, the file is created, and the OPEN is SUCCESSFUL.

General Rules OPEN EXTEND

1. A file that is OPEN EXTEND may successfully execute the UNLOCK, WRITE, and CLOSE statements.
2. A file that is OPEN EXTEND may not perform READ statements, or make updates to the file via the DELETE, or REWRITE statement.
3. A file that has successfully executed an OPEN EXTEND statement positions the file pointer after the last record.
4. If the file does not exist when the OPEN Extend statement is executed, then the OPEN

statement fails with a FILE STATUS of 35 unless the SELECT phrase of the file contains the OPTIONAL phrase. In this case, the file is created, and the OPEN is SUCCESSFUL.

Code Samples

```
OPEN OUTPUT SEQ-FILE-1.
OPEN INPUT  SEQ-FILE-1.
OPEN EXTEND SEQ-FILE-1.

OPEN OUTPUT REL-FILE-1.
OPEN INPUT  REL-FILE-1.
OPEN I-O    REL-FILE-1.

OPEN OUTPUT IDX-FILE-1.
OPEN INPUT  IDX-FILE-1.
OPEN I-O    IDX-FILE-1.

OPEN OUTPUT SHARING WITH NO OTHER IDX-FILE-1.
OPEN INPUT  SHARING WITH READ ONLY IDX-FILE-1.

OPEN OUTPUT SEQ-FILE-1 WITH NO REWIND.
OPEN INPUT  SEQ-FILE-1 WITH NO REWIND.

OPEN INPUT  REL-FILE-1 WITH LOCK.
OPEN I-O    REL-FILE-1 WITH LOCK.

OPEN INPUT IDX-FILE-1 WITH LOCK.
OPEN I-O IDX-FILE-1 WITH LOCK.
```

5.3.30 PERFORM Statement

The PERFORM statement executes a series of statements iteratively.

Format 1

A Format 1 PERFORM statement executes a procedure, or the code beginning in a proc-1 up to, and including the code in a named proc-2 once, or multiple times as designated by the FOREVER phrase or the TIMES phrase.

```
PERFORM [proc-1 [ THRU proc-2 ] ]
        [ { FOREVER         } ]
          {numeric-1 TIMES }
```

Syntax

1. proc-n is a procedure or section declared in the program.
2. numeric-n is a numeric literal or data item.
3. num-n is a numeric literal or data item.

General Rules

1. THRU and THROUGH are synonyms

2. When the THRU statement is not used, the scope of the Format 1 PERFORM statement is all of the statements within proc-1.
3. When the THRU statement is used, the scope of the Format 1 PERFORM statement is all of the statements within proc-1, all of the statements between proc-1 and proc-2, and all of the statements in proc-2.
4. In a proc-1 thru proc-2 construct, proc-2 must follow proc-1 in the source code.
5. The TIMES phrase indicates that the number of times that statements within the scope of the PERFORM statement are to be executed. After the statements within the scope of the PERFORM statement have been executed the designated number of TIMES, control returns to the next statement in the program after the PERFORM statement.
6. The FOREVER phrase indicates that the statements within the scope of the PERFORM statement should continue to execute continuously. A program may exit from a PERFORM FOREVER phrase with an EXIT PERFORM CYCLE statement.

Format 2

A Format 2 PERFORM statement executes a procedure, or the code beginning in a proc-1 up to, and including the code in a named proc-2 UNTIL condition-1 tests true.

```
PERFORM [proc-1 [ THRU proc-2 ] ]
       [WITH TEST {BEFORE}]
                  {AFTER }
       [VARYING { num-1 FROM num-2 by num-3 } ] UNTIL condition-1
```

Syntax

1. proc-n is a procedure or section declared in the program.
2. num-n is a numeric literal or data item.
3. condition-1 is a conditional statement.

General Rules

1. THRU and THROUGH are synonyms.
2. When the THRU statement is not used, the scope of the Format 2 PERFORM statement is all of the statements within proc-1.
3. When the THRU statement is used, the scope of the Format 2 PERFORM statement is all of the statements within proc-1, all of the statements between proc-1 and proc-2, and all of the statements in proc-2.
4. In a proc-1 THRU proc-2 construct, proc-2 must follow proc-1 in the source code.
5. In a Format 2 PERFORM statement, the WITH TEST phrase is optional. In the absence of a WITH TEST phrase, WITH TEST BEFORE is assumed.
6. The WITH TEST BEFORE phrase indicates that the test of condition-1 takes place before executing the statements within the scope of the PERFORM statement. If condition-1 tests TRUE initially, the statements within the scope of the PERFORM statement are not executed, and control is passed to the next statement after the PERFORM statement.
7. The WITH TEST AFTER phrase indicates that the test of condition-1 takes place after executing the statements within the scope of the PERFORM statement. When the WITH TEST AFTER phrase is used, the statements within the scope of the PERFORM statement will be executed at least one time.

8. The VARYING phrase is used to increment, or decrement a numeric counter variable by a fixed amount. When using the VARYING phrase , condition-1 is set to a test of the numeric counter variable.
9. Condition-1 may contain the word EXIT. This is a PERFORM UNTIL EXIT statement, and the condition tests true when an EXIT PERFORM CYCLE statement is reached.

Format 3

A Format 3 PERFORM statement executes a statement-list, iterated within the bounds of a PERFORM and END-PERFORM statement, once, or multiple times as designated by the FOREVER phrase or the TIMES phrase.

```
PERFORM [ { FOREVER    } ]
          {numeric-1 TIMES }
        [ statement-list ]
        [ END-PERFORM ]
```

Syntax

1. numeric-n is a numeric literal or data item.
2. statement-list is a list of imperative and/or conditional statements.

General Rules

1. The scope of the Format 3 PERFORM statement is all of the statements in statement-list, which are entered between the PERFORM and END-PERFORM statements.
2. The TIMES phrase indicates that the number of times that statements within the scope of the PERFORM statement are to be executed. After the statements within the scope of the PERFORM statement have been executed the designated number of TIMES, control returns to the next statement in the program after the PERFORM statement.
3. The FOREVER phrase indicates that the statements within the scope of the PERFORM statement should continue to execute continuously. A program may exit from a PERFORM FOREVER phrase with an EXIT PERFORM CYCLE statement.

Format 4

A Format 4 PERFORM statement executes a statement-list, iterated within the bounds of a PERFORM and END-PERFORM statement UNTIL a named condition tests true.

```
PERFORM
        [WITH TEST {BEFORE}]
                   {AFTER }
         VARYING { num-1 FROM num-2 by num-3 } ] UNTIL condition-1
```

```
[ statement-list ]
[ END-PERFORM ]
```

Syntax

1. num-n is a numeric literal or data item.
2. condition-1 is a conditional statement.
3. statement-list is a list of imperative and/or conditional statements.

General Rules

1. The scope of the Format 4 PERFORM statement is all of the statements in statement-list, which are entered between the PERFORM and END-PERFORM statements.
2. Condition-1 is the condition-test that determines whether or not to perform the statements within the scope of the PERFORM statement.
3. Condition-1 may contain the word EXIT. This is a PERFORM UNTIL EXIT statement, and the condition tests true when an EXIT PERFORM CYCLE statement is reached.
4. In a Format 4 PERFORM statement, the WITH TEST phrase is optional. In the absence of a WITH TEST phrase, WITH TEST BEFORE is assumed.
5. The WITH TEST BEFORE phrase indicates that the test of condition-1 takes place before executing the statements within the scope of the PERFORM statement. If condition-1 tests TRUE initially, the statements within the scope of the PERFORM statement are not executed, and control is passed to the next statement after the PERFORM statement.
6. The WITH TEST AFTER phrase indicates that the test of condition-1 takes place after executing the statements within the scope of the PERFORM statement. When the WITH TEST AFTER phrase is used, the statements within the scope of the PERFORM statement will be executed at least one time.
7. The VARYING phrase is used to increment, or decrement a numeric counter variable by a fixed amount. When using the VARYING phrase , condition-1 is set to a test of the numeric counter variable.

Code Sample

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PERFORM-1.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 CTR1    PIC 9(4).
77 CTR2    PIC 9(4).
77 CTR3    PIC 9(4).
77 CTR4    PIC 9(4).
77 COUNTER PIC 9(4).
77 DUMMY   PIC X.
*
PROCEDURE DIVISION.
MAIN.
```

```
           PERFORM PARA-1 WITH TEST BEFORE
             VARYING CTR1 FROM 1 BY 1 UNTIL CTR1 > 2.

           PERFORM PARA-1 THRU PARA-1-END WITH TEST AFTER
             VARYING CTR2 FROM 10 BY 10 UNTIL CTR2 > 20.

           MOVE 1 TO COUNTER.
           PERFORM FOREVER
             ADD 1 TO COUNTER
             IF COUNTER > 2
               EXIT PERFORM
             END-IF
           END-PERFORM.


           MOVE 1 TO COUNTER.
           PERFORM 1 TIMES
             ADD 1 TO COUNTER
             IF COUNTER > 2
               EXIT PERFORM
             END-IF
           END-PERFORM.

           MOVE 1 TO COUNTER.
           PERFORM WITH TEST BEFORE
             VARYING CTR3 FROM 1 BY 1 UNTIL CTR3 > 2
             ADD 1 TO COUNTER
             IF COUNTER > 2
               EXIT PERFORM CYCLE
             END-IF
           END-PERFORM.

           PERFORM WITH TEST AFTER
             VARYING CTR4 FROM 1 BY 1 UNTIL CTR4 > 2
             ADD 1 TO COUNTER
             IF COUNTER > 2
               EXIT PERFORM CYCLE
             END-IF
           END-PERFORM.


           PERFORM PARA-1 2 TIMES.

           PERFORM PARA-1 THRU PARA-1-END 2 TIMES.

           PERFORM PARA-2 THRU PARA-2-END FOREVER.
      *
       PARA-1.

           CONTINUE.
       PARA-1-END.
           EXIT.
       PARA-2.
           ADD 1 TO COUNTER.
           IF COUNTER >5
             DISPLAY "PERFORM-1 FINISHED!" LINE 15 COL 10
             ACCEPT DUMMY LINE 15 COL 30
             STOP RUN.
       PARA-2-END.
           EXIT.
```

5.3.31 PRAGMA Statement

The PRAGMA statement provides internal compiler control, for profiling. The first literal is the command sent to the compiler.

Format 1

```
PRAGMA  "PROFILING" { literal-1 } ...
```

Format 2

```
PRAGMA  "DUMP" { literal-1 } ...
```

Syntax

1. literal-n is a character string.

General Rules

1. The PRAGMA "PROFILING" statement causes programs compiled with –fprofiling to report time measurements from the current PRAGMA statement to the next PRAGMA , or to the end of the program.
2. The PRAGMA "PROFILING" statement should be entered in column 8, as in the Code Sample below.
3. The PRAGMA "DUMP" statement causes a profiling report to be generated when executed. Profiling reports produced by the PRAGMA "DUMP" statement will overwrite previous PRAGMA "DUMP" files with the same name.

PRAGMA "PROFILING" Code Sample

```
       PROCEDURE DIVISION.
       . . .
       PRAGMA "PROFILING" "STEP1".
       . . .
       PRAGMA "PROFILING" "STEP2".
```

PRAGMA "DUMP" Code Sample

```
       PROCEDURE DIVISION.
       . . .
       PRAGMA "DUMP" "REPORT".
       . . .
```

5.3.32 READ Statement

The READ statement retrieves records from files.

Format 1
A Format 1 Read is a Sequential READ, which retrieves the NEXT or PREVIOUS record, as determined by the current file pointer.

```
READ file-1     {NEXT    } RECORD INTO data-1
                {PREVIOUS}
                [{IGNORING LOCK    }]
                 {WITH LOCK        }
                 {WITH NO LOCK     }
                 {WITH IGNORE LOCK }
                 {WITH WAIT        }
                 {WITH KEPT LOCK   }
                [ AT END statement-1 ]
                [ NOT AT END statement-2 ]
                [ END-READ ]
```

1. file-n is a file described in the File Section with an FD.
2. data-n is a data item.
3. key-name-n is a data element that has been named as a key for an indexed file.
4. statement-n is an imperative statement.

General Rules

1. file-1 must be OPEN Input or I-O before the READ statement is executed.
2. The Format 1 READ statement applies to files declared with ACCESS MODE IS SEQUENTIAL or ACCESS MODE IS DYNAMIC.
3. A successful READ statement retrieves a record from an indexed, relative, or sequential file, and stores the record in the data area described in the File Description (FD) in the File Section of the program.
4. The NEXT and PREVIOUS phrases indicate in which direction the READ is to move the file pointer in the sequential READ operation.
5. The file pointer is initially placed at the beginning of the file after a successful OPEN statement.
6. In files described with ORGANIZATION IS SEQUENTIAL, a READ NEXT statement after an OPEN statement retrieves the first record in the file.
7. In files described with ORGANIZATION IS SEQUENTIAL, the READ NEXT and READ PREVIOUS statements retrieve the next/previous physical record in the file after/before the current position of the file pointer.
8. In files described with ORGANIZATION IS RELATIVE, the READ NEXT statement retrieves the next physical record in the file after the position marked by the RELATIVE KEY, and increments the relative-key data field. The READ PREVIOUS statement retrieves the previous physical record in the file after the position marked by the RELATIVE KEY and decrements the relative-key data field.
9. In files described with ORGANIZATION IS INDEXED, the READ NEXT and READ

PREVIOUS statements retrieve the next/previous physical record in the file after/before the current position of the file pointer.

10. The file pointer may be placed anywhere in the file with the START statement. For details, see START statement.
11. The INTO phrase causes a MOVE to data-1 after the data has been retrieved in the record area of the File Description (FD).
12. The IGNORING LOCK phrase ignores locks held by other programs.
13. IGNORING LOCK and WITH IGNORE LOCK are synonyms.
14. The WITH LOCK phrase causes the record to be locked. A locked record cannot be READ WITH LOCK or updated by another program.
15. WITH LOCK and WITH KEPT LOCK are synonyms.
16. The WITH NO LOCK phrase allows access to the record by other programs.
17. The WITH IGNORE LOCK phrase ignores locks held by other programs.
18. The WITH WAIT phrase causes the READ statement to wait for a lock held on record to be released.
19. The AT END condition is triggered when the end of the file is reached in a sequence of READ NEXT statements, or when the beginning of the file is reached in a sequence of READ PREVIOUS statements.
20. If the AT END condition is triggered, and the AT END phrase is included in the READ statement, the FILE STATUS variable is set to "10", the statement-list within the scope of the AT END phrase is executed, and control is then passed to the next statement after the READ.
21. The NOT AT END condition is triggered when the sequential READ is successful. When the NOT AT END phrase is included in the READ statement, and the READ is successful, the statement-list within the scope of the NOT AT END phrase is executed, and control is then passed to the next statement after the READ.

Format 2

A Format 2 READ is a KEY'ed READ, which READs on the relative key of a relative file, or one of the record keys of an indexed file.

```
READ file-1     RECORD INTO data-1
                [{IGNORING LOCK    }]
                 {WITH LOCK        }
                 {WITH NO LOCK     }
                 {WITH IGNORE LOCK }
                 {WITH WAIT        }
                 {WITH KEPT LOCK   }
                 [ KEY IS { data-2 } ]
                          { key-name-1 } ]
                 [ INVALID KEY statement-3 ]
                 [ NOT INVALID KEY statement-4 ]
                 [ END-READ ]
```

Syntax

1. file-n is a file described in the File Section with an FD.

2. data-n is a data item.
3. key-name-n is a data element that has been named as a key for an indexed file.
4. statement-n is an imperative statement.

General Rules

1. file-1 must be OPEN Input or I-O before the READ statement is executed.
2. The Format 2 READ statement applies to files declared with ACCESS MODE IS RANDOM or ACCESS MODE IS DYNAMIC.
3. The INTO phrase causes a MOVE to data-1 after the data has been retrieved in the record area of the File Description (FD).
4. The KEY clause is optional.
5. If the KEY clause is not present, then:
    a. In relative files, the value of the relative key field is used as the key
    b. In indexed files, the current value of the primary key stored in the record in the File Sectio is used as the key.
6. The KEY clause, when used, references keys as follows:
    a. In relative files, the value of the data field referenced is used as the relative key.
    b. In indexed files, the data field referenced must identify either the primary key or one of the alternate record keys.
7. A successful READ statement retrieves a record from an indexed, or relative file, and stores the record in the data area described in the File Description (FD) in the File Section of the program.
8. When a READ statement is made using an alternate key that allows duplicates, the first record that matches the given key is retrieved.
9. The INTO phrase causes a MOVE to data-1 after the data has been retrieved in the record area of the File Description (FD).
10. The WITH LOCK phrase causes the record to be locked. A locked record cannot be READ WITH LOCK or updated by another program.
11. WITH LOCK and WITH KEPT LOCK are synonyms.
12. The IGNORING LOCK phrase ignores locks held by other programs.
13. IGNORING LOCK and WITH IGNORE LOCK are synonyms.
14. The WITH NO LOCK phrase allows access to the record by other programs.
15. The WITH WAIT phrase causes the READ statement to wait for a lock held on record to be released.
16. The INVALID KEY condition is triggered in a READ on a relative file when a RELATIVE KEY data field does not contain a positive integer value.
17. The INVALID KEY condition is triggered in a READ on an indexed file when the RECORD KEY is not found.
18. When the INVALID KEY condition is triggered, and the INVALID KEY clause is included in the READ statement, the FILE STATUS variable is set to “23”, the statement-list within the scope of the INVALID KEY phrase is executed, and control is then passed to the next statement after the READ.
19. The NOT INVALID KEY condition is triggered when the READ is successful. When the NOT INVALID KEY phrase is included in the READ statement, and the READ is

successful, the statement-list within the scope of the NOT INVALID KEY phrase is executed, and control is then passed to the next statement after the READ.

20. The READ statement updates the FILE STATUS variable.
21. If a READ statement is unsuccessful, the file pointer is undefined.

Code Sample

```
...
       SELECT SEQ-FILE-1 ASSIGNTO"SEQ-FILE-1"
       ORGANIZATION IS SEQUENTIAL
       ACCESS IS SEQUENTIAL
       FILE STATUS IS SEQ-FILE-1-STAT.
...
   FD SEQ-FILE-1.
   01 SEQ-FILE-1-RECORD    PIC X(10).
...
   77 SEQ-FILE-1-STATPIC XX.
   01 WS-SEQ-FILEPIC X(10).
...
       READ SEQ-FILE-1 NEXT RECORD.
       READ SEQ-FILE-1 NEXT RECORD INTO WS-SEQ-FILE.
       READ SEQ-FILE-1 NEXT RECORD IGNORING LOCK.
       READ SEQ-FILE-1 NEXT RECORD WITH LOCK.
       READ SEQ-FILE-1 NEXT RECORD WITH NO LOCK.
       READ SEQ-FILE-1 NEXT RECORD WITH IGNORE LOCK.
       READ SEQ-FILE-1 NEXT RECORD WITH WAIT.
       READ SEQ-FILE-1 NEXT RECORD WITH KEPT LOCK.
       READ SEQ-FILE-1 NEXT RECORD
         AT END MOVE "10" TO SEQ-FILE-1-STAT
         NOT AT END
           CONTINUE
       END-READ.

       SELECT IDX-FILE-1 ASSIGN TO "IDX-FILE-1"
       ORGANIZATION IS INDEXED
       ACCESS IS DYNAMIC
       RECORD KEY IS IDX-KEY
       FILE STATUS IS IDX-FILE-1-STAT.


   FD IDX-FILE-1.
   01 IDX-FILE-1-RECORD.
     03 IDX-KEY PIC X(10).

   77 IDX-FILE-1-STAT PIC XX.
   01 WS-IDX-FILE PIC X(10).

 ...
       READ IDX-FILE-1 NEXT RECORD.
       READ IDX-FILE-1 NEXT RECORD INTO WS-IDX-FILE.
       READ IDX-FILE-1 PREVIOUS RECORD.
       READ IDX-FILE-1 IGNORING LOCK.
       READ IDX-FILE-1 WITH LOCK.
       READ IDX-FILE-1 WITH NOLOCK.
       READ IDX-FILE-1 WITH IGNORE LOCK.
       READ IDX-FILE-1 WITH WAIT.
       READ IDX-FILE-1 WITH KEPT LOCK.
       READ IDX-FILE-1
         INVALID KEY MOVE "23" TO IDX-FILE-1-STAT
         NOT INVALID KEY
           CONTINUE
```

```
            END-READ.
```

5.3.33 READY/RESET TRACE Statements

The READY/RESET TRACE statements enable paragraph tracing in a program compiled with the -fready-trace compiler flag.

General Format

```
READY TRACE
. . .
RESET TRACE
```

General Rules

1. The READY TRACE and RESET TRACE statements enable paragraph tracing and are otherwise ignored in the execution of the program.
2. Paragraphs/Sections entered in the interval between the READY TRACE and RESET TRACE statement are traced by writing output to the console in the form:

PROGRAM-ID: [program-id]: [paragraph name]

In the example below:
PROGRAM-ID: READYTRACE: INITIALIZATIONS
PROGRAM-ID: READYTRACE: MAIN-BODY

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. READYTRACE.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       PROCEDURE DIVISION.
       MAIN.
           READY TRACE.
           PERFORM INITIALIZATIONS.
           PERFORM MAIN-BODY.
           RESET TRACE.
           STOP RUN.
           ….
```

5.3.34 RELEASE Statement

The RELEASE statement makes records available for a SORT in an INPUT PROCEDURE.

General Format

```
RELEASE sort-record-1 [ FROM identifier-1 ]
```

Syntax

1. sort-record-n is a record described in a Sort Description (SD) in the File Section.
2. identifier-n is a data element, literal, or data returned from a function call.

General Rules

3. The RELEASE statement can only be used in the INPUT PROCEDURE of a SORT statement.
4. The RELEASE statement MOVEs data from identifier-1 to a sort-record, which is declared in a Sort Description (SD) in the FILE SECTION of the program.
5. The RELEASE statement adds the data MOVE'd from identifier-1 to the SORT file.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. RELEASE-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
      *SORT INPUT FILE
           SELECT MEMBER-FILE ASSIGN TO "MEMBERSHIP.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERSHIP-STAT.

      *SORT OUTPUT FILE
           SELECT MEMBER-LIST ASSIGN TO "MEMBERLIST.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERLIST-STAT.

      *SORTFILE (SD)
           SELECT MEMBER-SORT ASSIGN TO "SORT-WORK".

       DATA DIVISION.
       FILE SECTION.
       FD MEMBER-FILE.
       01 MEMBER-INFO PIC X(40).

       FD MEMBER-LIST.
       01 SORTED-MEMBER-INFOPIC X(40).

       SD MEMBER-SORT.
       01 SORT-DATA.
         05 MEMBER-FIRST-NAME PIC X(10).
```

```
          05 MEMBER-LAST-NAME  PIC X(20).
          05 YEAR-JOINED       PIC X(4).
          05 MEMBER-RANK       PIC X(6).

       WORKING-STORAGE SECTION.
       77 MEMBERLIST-STAT PIC XX.
         88 EOF-MEMBERLIST  VALUE "10".
       77 MEMBERSHIP-STAT PIC XX.
         88 EOF-MEMBERSHIP  VALUE "10".
       77 OUTPUT-SORT-AT-END  PIC X.
         88 EOF-SORT-FILE VALUE "Y".
       77 DUMMY               PIC X.

       PROCEDURE DIVISION.
       MEMBER-SORT-PROC.

           SORT MEMBER-SORT
             ON ASCENDING KEY MEMBER-LAST-NAME
             ON DESCENDING KEY MEMBER-RANK
             WITH DUPLICATES IN ORDER
             INPUT PROCEDURE INPUT-PROC
             OUTPUT PROCEDURE OUTPUT-PROC.

           DISPLAY "SORT-3 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.
           STOP RUN.


       INPUT-PROC SECTION.
       INPUT-PROCESS.

           OPEN INPUT MEMBER-FILE.
           READ MEMBER-FILE NEXT RECORD.

           PERFORM UNTIL EOF-MEMBERLIST
             RELEASE SORT-DATA FROM MEMBER-INFO

             READ MEMBER-FILE NEXT RECORD
               AT END MOVE "10" TO MEMBERLIST-STAT
             END-READ
           END-PERFORM.

           CLOSE MEMBER-FILE.
           EXIT SECTION.

       OUTPUT-PROC SECTION.
       OUTPUT-PROCESS.
           OPEN OUTPUT MEMBER-LIST.
           PERFORM UNTIL EOF-SORT-FILE
             RETURN MEMBER-SORT
               AT END MOVE "Y" TO OUTPUT-SORT-AT-END
               NOT AT END
                 WRITE SORTED-MEMBER-INFO FROM SORT-DATA
             END-RETURN
           END-PERFORM.

           CLOSE MEMBER-LIST.
           EXIT SECTION.
```

5.3.35 RETURN Statement

The RETURN statement returns records from SORT or MERGE operations to the operating program from within an OUTPUT procedure.

General Format

```
RETURN sort-file-1 RECORD [ INTO data-1 ]
   AT END statement-1
   [NOT AT END statement-2 ]
   [END-RETURN ]
```

Syntax

1. sort-file-n is a file described in the File Section with an SD.
2. data-n is a data item.
3. statement-n is an imperative statement.

General Rules

1. The RETURN statement can only be used in the OUTPUT PROCEDURE of a SORT or MERGE statement.
2. The INTO phrase causes the data to be MOVE'd to data-1.
3. If there is no INTO phrase, the data is returned into the sort-record area in the sort file.
4. When the end of a sort file is reached, the AT END condition is triggered, otherwise the NOT AT END condition is triggered.
5. The RETURN statement does not update the File Status variable.

An excerpt from the sample program illustrates the use of the RETURN statement in an OUTPUT PROCEDURE. In the example below, the program RETURNs records from the sort file, writes them out to a new file.

```
            OPEN OUTPUT MEMBER-LIST.
            PERFORM UNTIL EOF-SORT-FILE
             RETURN MEMBER-SORT
               AT END MOVE "Y" TO OUTPUT-SORT-AT-END
               NOT AT END
                 WRITE SORTED-MEMBER-INFO FROM SORT-DATA
             END-RETURN
            END-PERFORM.
```

Code Sample

```
       IDENTIFICATION DIVISION.
```

```
       PROGRAM-ID. RETURN-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
      *SORT INPUT FILE
           SELECT MEMBER-FILE ASSIGN TO "MEMBERSHIP.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERSHIP-STAT.

      *SORT OUTPUT FILE
           SELECT MEMBER-LIST ASSIGN TO "MEMBERLIST.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERLIST-STAT.

      *SORTFILE (SD)
           SELECT MEMBER-SORT ASSIGN TO "SORT-WORK".

       DATA DIVISION.
       FILE SECTION.
       FD MEMBER-FILE.
       01 MEMBER-INFOPIC X(40).

       FD MEMBER-LIST.
       01 SORTED-MEMBER-INFO PIC X(40).

       SD MEMBER-SORT.
       01 SORT-DATA.
         05 MEMBER-FIRST-NAME PIC X(10).
         05 MEMBER-LAST-NAME  PIC X(20).
         05 YEAR-JOINED       PIC X(4).
         05 MEMBER-RANK       PIC X(6).

       WORKING-STORAGE SECTION.
       77 MEMBERLIST-STAT PIC XX.
         88 EOF-MEMBERLIST  VALUE "10".
       77 MEMBERSHIP-STAT PIC XX.
         88 EOF-MEMBERSHIP  VALUE "10".
       77 OUTPUT-SORT-AT-END  PIC X.
         88 EOF-SORT-FILEVALUE "Y".
       77 DUMMY               PIC X.

       PROCEDURE DIVISION.
       MEMBER-SORT-PROC.

           SORT MEMBER-SORT
             ON ASCENDING KEY MEMBER-LAST-NAME
             ON DESCENDING KEY MEMBER-RANK
             WITH DUPLICATES IN ORDER
             INPUT PROCEDURE INPUT-PROC
             OUTPUT PROCEDURE OUTPUT-PROC.

           DISPLAY "SORT-3 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.
           STOP RUN.


       INPUT-PROC SECTION.
       INPUT-PROCESS.

           OPEN INPUT MEMBER-FILE.
           READ MEMBER-FILE NEXT RECORD.
```

```
           PERFORM UNTIL EOF-MEMBERLIST
             RELEASE SORT-DATA FROM MEMBER-INFO

             READ MEMBER-FILE NEXT RECORD
               AT END MOVE "10" TO MEMBERLIST-STAT
             END-READ
           END-PERFORM.

           CLOSE MEMBER-FILE.
           EXIT SECTION.

       OUTPUT-PROC SECTION.
       OUTPUT-PROCESS.
           OPEN OUTPUT MEMBER-LIST.
           PERFORM UNTIL EOF-SORT-FILE
             RETURN MEMBER-SORT
               AT END MOVE "Y" TO OUTPUT-SORT-AT-END
               NOT AT END
                 WRITE SORTED-MEMBER-INFO FROM SORT-DATA
             END-RETURN
           END-PERFORM.

           CLOSE MEMBER-LIST.
           EXIT SECTION.
```

5.3.36 REWRITE Statement

The REWRITE statement modifies a record in an indexed file, posting modifications to non-key fields.

General Format

```
REWRITE record-1 [ FROM identifier-1 ]
     [ WITH [NO] LOCK  ]
     [ INVALID KEY statement-1 ]
     [ NOT INVALID KEY statement-2 ]
     [ END-REWRITE ]
```

Syntax

1. record-n is the name of a record declared in the FILE Section.
2. identifier-n is a data element, literal, or data returned from a function call.
3. statement-n is an imperative statement.

General Rules

1. The REWRITE statement can only be issued on records in a file that is OPEN I-O.
2. The REWRITE statement allows non-key fields to be altered and re-written to a file that is

OPEN I-O.

3. For files declared with ACCESS IS SEQUENTIAL, the only valid target of a REWRITE statement is the record most recently retrieved by a successful READ statement. The effect of the REWRITE statement is to replace the contents of the record with the new record.
4. For files declared with ACCESS IS RANDOM or ACCESS IS DYNAMIC, the target of a REWRITE statement is determined by the setting of the primary key in the case of indexed files, and the relative key in the case of relative files.
5. The REWRITE statement does not allow records to altered in length.
6. The FROM phrase causes a MOVE from identifier-1 to the record area of record-1 prior to the execution of the REWRITE statement.
7. The WITH [NO] LOCK phrase is optional.
8. The INVALID KEY/NOT INVALID KEY phrase requires that the file be of ORGANIZATION RELATIVE or ORGANIZATION INDEXED.
9. The INVALID KEY condition is triggered by any of the following:
    a. A change has been made to an indexed file's primary key.
    b. A changes has been made to an alternate key that does not allow duplicates, and the change has created a duplicate key condition.
    c. The size of the record has changed
    d. The INVALID KEY condition causes the REWRITE to fail. If an INVALID KEY phrase is used in the REWRITE statement, then the following statement-list is executed. If no INVALID KEY condition is used, then the condition may be detected in the DECLARATIVES. If there are no DECLARATIVES, then the program aborts.
10. The NOT INVALID KEY condition exists if the INVALID KEY condition is not triggered. If a NOT INVALID KEY phrase is used in the REWRITE statement, then the following statement-list is executed.
    a. The REWRITE statement updates the FILE STATUS variable.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. REWRITE-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT RESWORDS ASSIGN TO "RESWORDS"
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS RESERVED-WORD
           FILE STATUS IS RESWORDS-STAT.

       DATA DIVISION.
       FILE SECTION.
       FD RESWORDS.
       01 RESWORDS-RECORD.
         03 RESERVED-WORD         PIC X(30).
         03 COMMENT               PIC X(20).

       WORKING-STORAGE SECTION.
       77 RESWORDS-STAT PIC XX.
         88 END-OF-RESWORDS  VALUE "10".
       77 DUMMY                   PIC X.
```

```
       PROCEDURE DIVISION.
       MAIN.
           OPEN OUTPUT RESWORDS.
           MOVE "ACCEPT" TO RESERVED-WORD.
           MOVE "HELLO WORLD" TO COMMENT.
           WRITE RESWORDS-RECORD.
           CLOSE RESWORDS.

           OPEN I-O RESWORDS.
           MOVE"ACCEPT"TO RESERVED-WORD.
           READ RESWORDS.
           MOVE "CHANGING COMMENT" TO COMMENT.
           REWRITE RESWORDS-RECORD WITH LOCK
             INVALID KEY
               DISPLAY "REWRITE FAILED! " LINE 10 COL 10
               DISPLAY RESWORDS-STAT LINE 10 COL 26
             NOT INVALID KEY
               DISPLAY "REWRITE SUCCEEDS!" LINE 10 COL 10
           END-REWRITE.

           DISPLAY "REWRITE-1 FINISHED!" LINE 11 COL 10.
           ACCEPT DUMMY LINE 11 COL 35.

           CLOSE RESWORDS.
           STOP RUN.
```

5.3.37 ROLLBACK Statement

The ROLLBACK statement causes a FILE I-O operation to be cancelled.

ROLLBACK is only available when underlying file system supports it.

General Format

```
ROLLBACK.
```

General Rules

1. The ROLLBACK statement causes unwritten file buffers to not be written to the target file.
2. After cancelling the unwritten FILE I-O operations, the ROLLBACK statement releases record and file locks held by the target file.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. ROLLBACK-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT RESWORDS ASSIGN TO "RESWORDS"
```

```
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS RESERVED-WORD
           FILE STATUS IS RESWORDS-STAT.

       DATA DIVISION.
       FILE SECTION.
       FD RESWORDS.
       01 RESWORDS-RECORD.
         03 RESERVED-WORD         PIC X(30).

       WORKING-STORAGE SECTION.
       77 RESWORDS-STATPIC XX.
         88 END-OF-RESWORDS  VALUE"10".
       77 DUMMY                  PIC X.

       PROCEDURE DIVISION.
       MAIN.
           OPEN OUTPUT RESWORDS.
           MOVE "ACCEPT" TO RESERVED-WORD.
           WRITE RESWORDS-RECORD.
           ROLLBACK.
           DISPLAY "ROLLBACK-1 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.

           CLOSE RESWORDS.
           STOP RUN.
```

5.3.38 SEARCH Statement

The SEARCH statement does a sequential or binary search of a table that is populated with data. The binary search (SEARCH ALL) statement requires that the data be pre-sorted.

Format 1

The Format 1 Search statement reads through a table sequentially until a condition tests TRUE, or until the end of the table is reached.

```
SEARCH search-table-1 [VARYING {index-1 }
    [ AT END statement-1 ]
    { WHEN condition-1 {statement-2   } } ...
                       {NEXT SENTENCE }
    [ END-SEARCH ]
```

Syntax

1. search-tbl-n is a data item with an occurs clause, and an index. When used with the SEARCH ALL, a key is also required.
2. index-n is a table index, and must be described with USAGE INDEX.
3. statement-n is an imperative statement.

General Rules

1. The VARYING clause in not necessary if the table being SEARCH'ed contains an INDEXED BY clause.
2. Index-1 defines the starting position in the table of the search. For a search of a table from beginning, to end, index-1 should be SET to 1 prior to the execution of the SEARCH statement, as follows:
   SET index-1 TO 1.
3. The WHEN condition-1 phrase is tested for each instance of the table during the SEARCH. When condition-1 tests TRUE, statement-2 is executed, after which the SEARCH is terminated, and control passes to the next statement after the SEARCH statement.
4. Where multiple WHEN condition phrases are listed consecutively, they will each be tested for each instance of the table during the SEARCH. When any WHEN condition phrase tests TRUE, the statement-list associated with it is executed, after which the SEARCH is terminated, and control passes to the next statement after the SEARCH statement.
5. When the condition-test is TRUE, the index stores the location in the table of the element that has been identified.
6. The AT END condition is triggered when the entire table is SEARCH'ed and no WHEN condition phrase tests true. When the AT END condition is triggered, the statement-list associated with the AT END phrase is executed, the SEARCH is terminated, and control passes to the next statement after the SEARCH statement.

Format 2

The Format 2 Search statement performs a binary search on a table that is sorted, and in which the sort-keys are described in the declaration of the table.

```
SEARCH ALL search-table-1
    [AT END statement-1 ]
    WHEN {data-1 {IS EQUAL TO } { identifier-3 }}
                 {IS =        } {literal-1      }
                 {arithmetic-expression} {condition-1}
    [AND {data-2 {IS EQUAL TO} { identifiier-4 } } ] ...
                 { IS =       } { literal-2       }
                 {arithmetic-expr   } { condition-name-2 |}
    {statement-2   }...
    {NEXT SENTENCE }
    [END-SEARCH ]
```

Syntax

1. search-tbl-n is a data item with an occurs clause, and an index. When used with the SEARCH ALL, a key is also required.
2. data-n is a data item.
3. identifier-n is a data element, literal, or data returned from a function call.
4. literal-n is a character string.
5. condition-n is a condition name described in a level 88 statement.

6. statement-n is an imperative statement.

General Rules

1. index-1 does not need to be SET before the execution of a SEARCH ALL statement.
2. The contents of a table that is the target of a SEARCH ALL statement must be pre-sorted. The manner in which the SEARCH ALL statement determines whether or not there is a match is to internally SET the index to a value that represents the midpoint of the table, and test the truth of the condition clause. After the condition test, the SEARCH ALL statement limits the rest of its SEARCH to either the data elements above, or below the midpoint in the table. In this manner, it performs a binary search, and determines whether or not there is a match to the condition clause.
3. The Format 2 SEARCH ALL statement only permits a single WHEN phrase.
4. In order to get predictable results, the WHEN phrase must identify a unique table entry.
5. A condition test may be made up of multiple condition clauses that are connected with AND or OR .
6. When the condition-test is TRUE, the index stores the location in the table of the element that has been identified.
7. The VARYING syntax is not relevant, and not allowed on a SEARCH ALL statement.
8. The AT END condition is triggered when the entire table is SEARCH'ed and no WHEN condition phrase tests true. When the AT END condition is triggered, the statement-list associated with the AT END phrase is executed, the SEARCH is terminated, and control passes to the next statement after the SEARCH statement.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  SEARCH-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ALLSTAR-TABLE.
         05 STAR-1  PIC X(30) VALUE "01HANK    AARON     BRAVES    ".
         05 STAR-2  PIC X(30) VALUE "02YOGI    BERRA     YANKEES   ".
         05 STAR-3  PIC X(30) VALUE "03RODNEY  CAREW     TWINS     ".
         05 STAR-4  PIC X(30) VALUE "04JOE     DIMAGGIO  YANKEES   ".
         05 STAR-5  PIC X(30) VALUE "05DENNIS  ECKERSLEY AS        ".
         05 STAR-6  PIC X(30) VALUE "06CARLTON FISK      RED SOX   ".
         05 STAR-7  PIC X(30) VALUE "07LOU     GEHRIG    YANKEES   ".
         05 STAR-8  PIC X(30) VALUE "08ROGERS  HORNSBY   CARDINALS ".
         05 STAR-9  PIC X(30) VALUE "09REGGIE  JACKSON   YANKEES   ".
         05 STAR-10 PIC X(30) VALUE "10SANDY   KOUFAX    DODGERS   ".
         05 STAR-11 PIC X(30) VALUE "11TOMMY   LASORDA   DODGERS   ".
         05 STAR-12 PIC X(30) VALUE "12EDDIE   MURRAY    ORIOLES   ".
         05 STAR-13 PIC X(30) VALUE "13PHIL    NIEKRO    BRAVES    ".
         05 STAR-14 PIC X(30) VALUE "14JIM     PALMER    ORIOLES   ".
         05 STAR-15 PIC X(30) VALUE "15BABE    RUTH      YANKEES   ".
         05 STAR-16 PIC X(30) VALUE "16TOM     SEAVER    METS      ".
         05 STAR-17 PIC X(30) VALUE "17BILL    VEECK     WHITE SOX ".
         05 STAR-18 PIC X(30) VALUE "19EARL    WEAVER    ORIOLES   ".
         05 STAR-19 PIC X(30) VALUE "19BILLY   WILLIAMS  CUBS      ".
         05 STAR-20 PIC X(30) VALUE "20CARL    YAZ       RED SOX   ".
      *
       01 ALLSTAR-TBL REDEFINES ALLSTAR-TABLE.
        05 STAR-TABLE OCCURS 20 TIMES
            ASCENDING KEY IS A-LAST-NAME
```

```
           INDEXED BY INDEX-1.
         10 A-NUMBER               PIC 99.
         10 A-FIRST-NAME           PIC X(8).
         10 A-LAST-NAME            PIC X(10).
         10 A-TEAM                 PIC X(10).

       77 DUMMY   PIC X.
       PROCEDURE DIVISION.
       MAIN.
      *  SEARCH VARYING REQUIRES THE INDEX BE SET BEFORE BEGINNING
           SET INDEX-1 TO1.

           SEARCH STAR-TABLE VARYING INDEX-1
             AT END
               DISPLAY "END OF SEARCH!" LINE 11 COL 10
                 WHEN A-LAST-NAME(INDEX-1) = "KOUFAX"
                   DISPLAY A-FIRST-NAME(INDEX-1) LINE10COL10
           END-SEARCH.

      *   SEARCH ALL REQUIRES THE TABLE BE SORTED ON A KEY FIELD
      *   NAMED IN THE DECLARATION.
      *   INDEX-1 IS AUTOMATICALLY INITIALIZED BY SEARCH ALL

           SEARCH ALL STAR-TABLE
             AT END
               DISPLAY "END OF SEARCH ALL!" LINE 13 COL 10
               WHEN A-LAST-NAME(INDEX-1) = "RUTH"
                 DISPLAY A-FIRST-NAME(INDEX-1) LINE 12 COL 10
            END-SEARCH.

            DISPLAY "SEARCH-1 FINISHED!" LINE 15 COL 10.
            ACCEPT DUMMY LINE 15 COL 30.
            STOP RUN.
```

5.3.39 SET Statement

The SET statement can be used to set data types and environment variables to prescribed values.

Format 1

The Format 1 SET statement sets the value of an environment variable. If the environment variable does not exist, it is created, and assigned the given value.

```
SET ENVIRONMENT { literal-1 TO identifier-1 } ...
```

Syntax

1. literal-n is a character string.
2. identifier-n is a data element, literal, or data returned from a function call.

General Rules

1. In the Format 1 SET statement, literal-1 is the name of an environment variable that is set to

the value of identifier-1.

2. The value of the environment variable can be retrieved by the ACCEPT... FROM ENVIRONMENT statement by the COBOL-IT program , and any subprograms called by the COBOL-IT program.

Code Sample

```
       77 COBOL-TYPE    PIC X(10) VALUESPACES.
       ...
           SET ENVIRONMENT "COBOL" TO "COBOL-IT".
           ACCEPT COBOL-TYPE FROM ENVIRONMENT "COBOL".
           DISPLAY COBOL-TYPE LINE 13 COL 10.
       ...
```

Format 2

The Format 2 SET statement sets a data item to a given value.

```
SET {data-1} ... TO identifier-1.
```

Syntax

1. data-n is a data item.
2. identifier-n is a data element, literal, or data returned from a function call.

General Rules

1. The Format 2 Set statement has the effect of a MOVE statement, assigning a given value to a data item.

   Code Sample

```
       77 NUMERIC-1    PIC 99 VALUE 0.
       77 ALPHA-1      PIC X(5) VALUE SPACES.
       ...
           SET NUMERIC-1 TO 10.
           DISPLAY NUMERIC-1 LINE 5 COL 10.
           SET ALPHA-1 TO "HELLO".
           DISPLAY ALPHA-1 LINE 5 COL 20.
       ...
```

Format 3

The Format 3 SET statement assigns data item with USAGE POINTER to the ADDRESS of a variable.

```
SET {data-ptr-1}... TO { ADDRESS OF identifier-1 }
                       { NULL                    }
```

Syntax

1. data-ptr-n is a data item declared with USAGE POINTER.
2. identifier-n is a data element, literal, or data returned from a function call.

General Rules

There are no General Rules.

Note: The Format 3 SET statement can be used in conjunction with the Format 4 SET statement to enable the use of a Linkage item in a sub-program which has not been part of a CALL USING statement.

For an example, see the Code Sample included with the Format 4 Set Statement.

Format 4

The Format 3 SET statement assigns data item with USAGE POINTER to the ADDRESS of a variable.

```
SET {ADDRESS OF linkage-data-item-1}... TO data-ptr-1
```

Syntax

1. linkage-data-item-n is a data item declared in the LINKAGE SECTION.

General Rules

There are no General Rules.

Note: The Format 3 SET statement can be used in conjunction with the Format 4 SET statement to enable the use of a Linkage item in a sub-program which has not been part of a CALL USING statement.

Referencing a Linkage item that had no address associated with it would normally abort a runtime session. In the sample below, the linkage item acquires a valid address, and can be referenced without a problem.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  SUBPGM.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 COBOL-TYPE   PIC X(8) VALUE "COBOL-IT".
       77 DATA-PTR-1   USAGE POINTER.
       77 DATA-PTR-2   USAGE POINTER.
       LINKAGE SECTION.
       01 LINK-1  PIC X(8).
       PROCEDURE DIVISION.
       MAIN.
           SET DATA-PTR-1 TO ADDRESS OF COBOL-TYPE.
           DISPLAY "DATA-PTR SET TO ADDRESS" LINE 6 COL 10.
           SET DATA-PTR-2 TO NULL.
           DISPLAY "DATA-PTR SET TO NULL" LINE 7 COL 10.
           SET ADDRESS OF LINK-1 TO DATA-PTR-1.
           DISPLAY "LINK-1: " LINE 8 COL 10.
           DISPLAY LINK-1 LINE 8 COL 20.
           GOBACK.
```

Format 5

```
SET {data-1} ... {UP   } BY integer-1
                 {DOWN}
```

Syntax

1. data-n is a data element that is an integer.
2. integer-n is a data element, literal or data returned from a function call that is an integer.

General Rules

1. The Format 5 SET statement has the effect of an ADD or SUBTRACT statement, incrementing or decrementing a numeric data item with an integer value be a given integer value.
2. The Format 5 SET statement may be used with indexes, as a means of directing table handling.

Code Sample

```
       77 NUMERIC-1  PIC 99 VALUE 20.
       ...
           SET NUMERIC-1 UP BY 10.
           DISPLAY NUMERIC-1 LINE 6 COL 10.
           SET NUMERIC-1 DOWN BY 5.
           DISPLAY NUMERIC-1 LINE 7 COL 10.
       ...
```

Format 6

```
SET { {condition-1} ... TO {TRUE } } ...
                           {FALSE}
```

Syntax

1. condition-n is a condition name that is described in a level 88 statement.

General Rules

1. The Format 6 SET statement has the effect of MOVE statement, MOVEing a value to the parent data item to create a TRUE condition when the condition is SET to TRUE, and MOVEing a value to the parent data item to create a FALSE condition when the condition is SET TO FALSE.
2. IF the level-88 condition contains a WHEN SET TO FALSE value, then the effect of the SET [condition-1] TO FALSE is to MOVE that value to the parent data item.
3. If an attempt is made to set a level-88 condition to FALSE, and that condition does not have an explicit WHEN SET TO FALSE clause, the compiler will generate an error.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  SET3.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 DECISION-FLAG PIC 9 VALUE 0.
         88 DECISION-YES   VALUES 1 THRU 5.
            WHEN SET TO FALSE 6.
       77 WS-DUMMY   PIC X.
       PROCEDURE DIVISION.
       MAIN.
           SET DECISION-YES TO TRUE.
           DISPLAY DECISION-FLAG LINE 8 COL 10.
           SET DECISION-YES TO FALSE.
           DISPLAY DECISION-FLAG LINE 9 COL 10.
           DISPLAY "ALL DONE!" LINE 10 COL 10.
           ACCEPT WS-DUMMY LINE 10 COL 30.
           STOP RUN.
```

Format 7

```
SET { {mnemonic-name-1} … TO (ON ) } …
                             (OFF)
```

Syntax

1. mnemonic-name is a user-defined word.

General Rules

1. The Format 7 SET statement SETs Switches that are described in SPECIAL-NAMES to ON or OFF. If the switch is described with an ON STATUS and OFF STATUS, these conditions can programmatically be checked for truth.

Code Sample

```
      ...
       SPECIAL-NAMES.
             SWITCH 1IS SWITCH-1
             ON STATUS IS SET-CHECK
             OFF STATUS IS SKIP-CHECK.
      ...

           SET SWITCH-1 TO ON.
      ...
           IF SET-CHECK
             DISPLAY "SWITCH 1 SET" LINE 4 COL 10
           END-IF.
      ...
```

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SET-1.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
           SWITCH 1IS SWITCH-1
           ON STATUS IS SET-CHECK
           OFF STATUS IS SKIP-CHECK.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 NUMERIC-1 PIC 99 VALUE 0.
       77 DECISION-FLAG PIC 9 VALUE 0.
         88 DECISION-YES VALUE 1
            WHEN SET TO FALSE 0.
       77 COBOL-TYPE PIC X(10) VALUE SPACES.
       77 DATA-PTR-1 USAGE POINTER.
       77 DATA-PTR-2 USAGE POINTER.
       01 DUMMY  PIC X.
```

```
       LINKAGE SECTION.
       01 LINKAGE-1         PIC X(10).
       PROCEDURE DIVISION.
       MAIN.
      * FORMAT 1
           SET ENVIRONMENT "COBOL" TO "COBOL-IT".
           ACCEPT COBOL-TYPE FROM ENVIRONMENT "COBOL".
           DISPLAY COBOL-TYPE LINE 4 COL 10.
      *
      * FORMAT 2
      *
           SET NUMERIC-1 TO 10.
           DISPLAY NUMERIC-1 LINE 5 COL 10.
      *
      * FORMAT 3
      *
           SET DATA-PTR-1 TO ADDRESS OF COBOL-TYPE.
           DISPLAY "DATA-PTR SET TO ADDRESS" LINE 6 COL 10.
           SET DATA-PTR-2 TO NULL.
           DISPLAY "DATA-PTR SET TO NULL" LINE 7 COL 10.
      *
      * FORMAT 4
      *
           SET ADDRESS OF LINKAGE-1 TO DATA-PTR-1.
           DISPLAY "ADDRESS OF LINKAGE ITEM SET" LINE 8 COL 10.
      *
      * FORMAT 5
      *
           SET NUMERIC-1 UP BY 10.
           DISPLAY NUMERIC-1 LINE 9 COL 10.
           SET NUMERIC-1 DOWN BY 5.
           DISPLAY NUMERIC-1 LINE 10 COL 10.
      *
      * FORMAT 6
      *
           SET DECISION-YES TO TRUE.
           DISPLAY DECISION-FLAG LINE 8 COL 10.
           SET DECISION-YES TO FALSE.
           DISPLAY DECISION-FLAG LINE 9 COL 10.
      *
      * FORMAT 7
      *
           SET SWITCH-1 TO ON.
           IF SET-CHECK
             DISPLAY "SWITCH 1 SET" LINE 11 COL 10
           END-IF.
      *
           DISPLAY "SET-1 FINISHED!" LINE 15 COL 10.
           ACCEPT DUMMY LINE 15 COL 30.

           STOP RUN.
```

5.3.40 SORT Statement

The SORT statement reorders the records in a file, or a table .

Format 1

The Format 1 SORT statement reorders the records in a file.

```
SORT sort-file-1
       { ON {ASCENDING } KEY {sort-key-1} } ...
         {DESCENDING}
       [ WITH DUPLICATES IN ORDER ]
       [ COLLATING SEQUENCE IS alphabet-name ]
       { USING {file-1} ...   }
       { GIVING {file-2} ... }
       { INPUT PROCEDURE IS proc-name  }
       { OUTPUT PROCEDURE IS proc-name }
```

Syntax

1. sort-file-n is a file described in the File Section with an SD.
2. sort-key-n is a data item that has been declared as a key to a sort table, or sort file.
3. alphabet-name is a user-defined word.
4. proc-name is the name of a paragraph or section in the program.
5. file-n is a file described in the File Section with an FD.

General Rules

1. Sort-file-1 must be described with an SD in the FILE SECTION.
2. The USING and GIVING files must be described with ORGANIZATION LINE SEQUENTIAL, or ORGANIZATION BINARY SEQENTIAL, and must be described with an FD in the FILE SECTION.
3. The order in which the KEY clauses are listed indicates the order in which the sort keys are applied. The first KEY listed is the primary key, the second key listed is the second key, etc...
4. The ASCENDING KEY clause causes values to sort from lowest to highest.
5. The DESCENDING KEY clause causes values to sort from highest to lowest.
6. The COLLATING SEQUENCE clause names the alphabet which is used for purposes of ordering KEY data in the SORT. Usage of the COLLATING SEQUENCE clause in a SORT statement overrides the naming of a PROGRAM COLLATING SEQUENCE in the object-computer paragraph.
7. WITH DUPLICATES IN ORDER is treated as commentary.
8. The SORT statement writes duplicates in the order in which they are encountered.
9. The file listed in the USING clause may not be OPEN when the SORT statement executes. The SORT statement OPENs the file.
10. The files listed in the GIVING clause may not be OPEN when the MERGE statement executes. The SORT statement OPENs the file.
11. The OUTPUT PROCEDURE is executed after all of the records have been SORTed into the temporary sort-file.
12. Inside the OUTPUT PROCEDURE, the RETURN statement is used to transfer records from the sort-file to the output file.
13. When the OUTPUT PROCEDURE is finished, the SORT statement closes all files.
14. The SORT statement updates file status variable that is defined for sort-file after performing

OPEN, READ, WRITE, and CLOSE operations. File Status errors may be detected in DECLARATIVES.

Format 2

The Format 2 SORT statement reorders the elements in a table.

```
SORT sort-tbl
       [ ON ASCENDING/DESCENDING KEY data-name-1 ... ]
       [ WITH DUPLICATES IN ORDER ]
       [ COLLATING SEQUENCE IS alphabet-name ]
```

Syntax

1. sort-tbl is a data- item with an occurs clause.
2. data-name-n is a data-item defined in the sort-tbl, subordinate to the OCCURS.
3. alphabet-name is a user-defined word.

General Rules

1. The WITH DUPLICATES IN ORDER phrase is treated as commentary.
2. The COLLATING SEQUENCE phrase is treated as commentary.
3. The Format 2 SORT statement does not require a sort file definition (SD) in the File Section.
4. The effect of the Format 2 SORT statement is a re-ordering of the elements in sort-tbl, as specified by the ASCENDING/DESCENDING KEY phrase(s).
5. The internal COBOL SORT checks for the setting of TMPDIR as the location for the storage of temporary SORT files. If TMPDIR is not set, then the behavior is OS-Dependent. In some operating systems, this can limit the size of files that can be SORTed by the internal COBOL SORT.

Code Sample

SORT USING/GIVING

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SORT-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
      *INPUT FILE
           SELECT MEMBER-FILE ASSIGN TO "MEMBERSHIP.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERSHIP-STAT.
```

```
      *OUTPUT FILE
           SELECT MEMBER-LIST ASSIGN TO "MEMBERLIST.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERLIST-STAT.

      *SORTFILE (SD)
           SELECT MEMBER-SORT ASSIGN TO "SORT-WORK".

       DATA DIVISION.
       FILE SECTION.
       FD MEMBER-FILE.
       01 MEMBER-INFOPIC X(40).

       FD MEMBER-LIST.
       01 SORTED-MEMBER-INFOPIC X(40).

       SD MEMBER-SORT.
       01 SORT-DATA.
        05 MEMBER-FIRST-NAME PIC X(10).
        05 MEMBER-LAST-NAME  PIC X(20).
        05 YEAR-JOINED       PIC X(4).
        05 MEMBER-RANK       PIC X(6).

       WORKING-STORAGE SECTION.
       77 MEMBERLIST-STAT PIC XX.
       77 MEMBERSHIP-STATPIC XX.

       PROCEDUREDIVISION.
       PRODUCT-LIST-SORT.
           SORT MEMBER-SORT
             ON ASCENDING KEY MEMBER-LAST-NAME
             ON DESCENDING KEY MEMBER-RANK
             WITH DUPLICATES IN ORDER
             USING  MEMBER-FILE
             GIVING MEMBER-LIST.
```

SORT Table

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SORT-2.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       77 MEMBERLIST-STAT PIC XX.
       77 MEMBERSHIP-STAT PIC XX.
       77 LINE-NUMBER     PIC 99 VALUE 10.
       77 CTR             PIC 99 VALUE 0.
       77 DUMMY           PIC X.

       01 MEMBER-TABLE.
         05 MEMBER-1 PIC X(20) VALUE "SAMUEL   JOHNSON    ".
         05 MEMBER-2 PIC X(20) VALUE "THOMAS   JEFFERSON  ".
         05 MEMBER-3 PIC X(20) VALUE "GEORGE   WASHINGTON ".
         05 MEMBER-4 PIC X(20) VALUE "SAMUEL   ADAMS      ".
         05 MEMBER-5 PIC X(20) VALUE "ABRAHAM  LINCOLN    ".
         05 MEMBER-6 PIC X(20) VALUE "WOODROW  WILSON     ".
```

```
          05 MEMBER-7 PIC X(20) VALUE "FRANKLIN ROOSEVELT  ".
          05 MEMBER-8 PIC X(20) VALUE "DWIGHT   EISENHOWER ".
        01 MEMBER-TABLE-RED REDEFINES MEMBER-TABLE.
          05 PATRIOT-TBL OCCURS 8 TIMES.
            10 FIRST-NAME PIC X(10).
            10 LAST-NAME  PIC X(10).
      *
       PROCEDURE DIVISION.
       MEMBER-SORT.

           SORT PATRIOT-TBL
             ON ASCENDING KEY FIRST-NAME.

           PERFORM VARYING CTR FROM 1 BY 1 UNTIL CTR > 8
             DISPLAY FIRST-NAME(CTR) LINE LINE-NUMBER COL 10
             DISPLAY LAST-NAME(CTR)  LINE LINE-NUMBER COL 20
             ADD 1 TO LINE-NUMBER
           END-PERFORM.


           DISPLAY "SORT-2 FINISHED!" LINE 20 COL 10.
           ACCEPT DUMMY LINE 20 COL 30.
           STOP RUN.
```

SORT INPUT/OUTPUT Procedures

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SORT-3.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
      *INPUT FILE
           SELECT MEMBER-FILE ASSIGN TO "MEMBERSHIP.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERSHIP-STAT.

      *OUTPUT FILE
           SELECT MEMBER-LIST ASSIGN TO "MEMBERLIST.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERLIST-STAT.

      *SORTFILE (SD)
           SELECT MEMBER-SORT ASSIGN TO "SORT-WORK".

       DATA DIVISION.
       FILE SECTION.
       FD MEMBER-FILE.
       01 MEMBER-INFO         PIC X(40).

       FD MEMBER-LIST.
       01 SORTED-MEMBER-INFO  PIC X(40).

       SD MEMBER-SORT.
       01 SORT-DATA.
         05 MEMBER-FIRST-NAME PIC X(10).
         05 MEMBER-LAST-NAME  PIC X(20).
         05 YEAR-JOINED       PIC X(4).
```

```
        05 MEMBER-RANK       PIC X(6).

      WORKING-STORAGESECTION.
      77 MEMBERLIST-STAT PIC XX.
        88 EOF-MEMBERLIST  VALUE "10".
      77 MEMBERSHIP-STAT PIC XX.
        88 EOF-MEMBERSHIP  VALUE "10".
      77 OUTPUT-SORT-AT-END  PIC X.
        88 EOF-SORT-FILE VALUE "Y".
      77 DUMMY               PIC X.

      PROCEDURE DIVISION.
      MEMBER-SORT-PROC.

          SORT MEMBER-SORT
            ON ASCENDING KEY MEMBER-LAST-NAME
            ON DESCENDING KEY MEMBER-RANK
            WITH DUPLICATES IN ORDER
            INPUT PROCEDURE INPUT-PROC
            OUTPUT PROCEDURE OUTPUT-PROC.

          DISPLAY "SORT-3 FINISHED!" LINE 10 COL 10.
          ACCEPT DUMMY LINE 10 COL 30.
          STOP RUN.

      INPUT-PROC SECTION.
      INPUT-PROCESS.
          OPEN INPUT MEMBER-FILE.
          READ MEMBER-FILE NEXT RECORD.

          PERFORM UNTIL EOF-MEMBERLIST
            RELEASE SORT-DATA FROM MEMBER-INFO

            READ MEMBER-FILE NEXT RECORD
              AT END MOVE "10" TO MEMBERLIST-STAT
            END-READ
          END-PERFORM.

          CLOSE MEMBER-FILE.
          EXIT SECTION.

      OUTPUT-PROC SECTION.
      OUTPUT-PROCESS.
          OPEN OUTPUT MEMBER-LIST.
          PERFORM UNTIL EOF-SORT-FILE
            RETURN MEMBER-SORT
              AT END MOVE "Y" TO OUTPUT-SORT-AT-END
              NOT AT END
                WRITE SORTED-MEMBER-INFO FROM SORT-DATA
            END-RETURN
          END-PERFORM.

          CLOSE MEMBER-LIST.
          EXIT SECTION.
```

5.3.41 START Statement

The START statement positions the file pointer of an indexed or relative file for a READ NEXT or READ PREVIOUS statement based on a condition that relates to the value of a KEY of the file.

General Format

```
START file-name-1
  [KEY IS { EQUAL TO }   keyname-1 ]
        [ { =              } ]
        [ { GREATER THAN   } ]
        [ { >              } ]
        [ { NOT LESS THAN } ]
                [ { NOT <          } ]
        [ { GREATER THAN OR EQUAL TO } ]
        [ { >=             } ]
        [ { LESS THAN      } ]
        [ { <              } ]
        [ { NOT GREATER THAN } ]
        [ { NOT >            } ]
        [ { <=               } ]
        [ { LESS THAN OR EQUAL TO } ]
        [ INVALID KEY statement-1 ]
        [ NOT INVALID KEY statement-2 ]
        [ END-START ]
```

Syntax

1. file-name-n is an indexed or relative file described in the FILE Section.
2. keyname-1 is a literal or data element whose value creates the condition test for the placing of the file pointer by the START statement.
3. statement-n is an imperative statement.

General Rules

1. file-name-1 must be OPEN INPUT or OPEN I-O when the START statement executes.
2. If file-name-1 is a relative file, the key of reference is the data item named by the RELATIVE KEY phrase in the SELECT clause of the file.
3. If file-name-1 is an indexed file, the key of reference may be either the data item(s) referenced in the RECORD KEY phrase, or the data item(s) referenced in an ALTERNATE RECORD KEY phrase.
4. If keyname-1 is the data name referenced by the RECORD KEY phrase, then the START/READ NEXT sequence will be made on the primary key.
5. If keyname-1 is the data name referenced by the ALTERNATE RECORD KEY phrase, then

the START/READ NEXT sequence will be made on the alternate record key.

6. A successful START statement positions the file pointer according to the following rules:
   a. Where the condition operand is one of the following: EQUAL (“ =”), GREATER THAN (“>”), GREATER THAN OR EQUAL (“>=”), the file pointer is placed at the first record that satisfies the KEY clause specification.
   b. Where the condition operand is one of the following: LESS THAN (“<”), LESS THEN OR EQUAL (“<=”), the file pointer is placed at the last record that satisfies the KEY clause specification.
7. The INVALID KEY condition exists if the START statement is unable to place the record pointer based on the condition described.
8. When the INVALID KEY condition exists, one of the following occurs:
9. If there is an INVALID KEY phrase in the START statement, the associated statement-list is executed, and control then passes to the next statement in the program.
10. If there is no INVALID KEY phrase in the START statement, the FILE STATUS variable of the file is updated with an error code, and control is either transferred to the DECLARATIVES, if they have been set up for this case, or the program aborts.
11. The NOT INVALID KEY condition exists if the START statement is able to plsce the record pointer based on the condition described.

Code Sample

```
...
       SELECT CUSTFILE ASSIGN TO "CUSTOMER"
       ORGANIZATION IS INDEXED
       ACCESS IS DYNAMIC
       RECORD KEY IS CUSTOMER-ID
       FILE STATUS IS CUSTOMER-STAT.
...
  FD CUSTFILE.
  01 CUSTOMER-RECORD.
    03 CUSTOMER-ID     PIC 9(5).
    03 CUSTOMER-NAME   PIC X(20).
    03 CUSTOMER-ADDR   PIC X(20).
    03 CUSTOMER-CITY   PIC X(10).
    03 CUSTOMER-STATE  PIC XX.
    03 CUSTOMER-PHONE  PIC X(10).
...
  77 CUSTOMER-STAT PIC XX.
...
       MOVE 11111 TO CUSTOMER-ID.

       START CUSTFILE KEY EQUAL TO CUSTOMER-ID.

       START CUSTFILE KEY = CUSTOMER-ID.

       START CUSTFILE KEY GREATER THAN CUSTOMER-ID.

       START CUSTFILE KEY > CUSTOMER-ID.

       START CUSTFILE KEY NOT LESS THAN CUSTOMER-ID.

       START CUSTFILE KEY NOT < CUSTOMER-ID.

       START CUSTFILE KEY GREATER THAN OR EQUAL TO CUSTOMER-ID.
```

```
START CUSTFILE KEY >= CUSTOMER-ID.

START CUSTFILE KEY LESS THAN CUSTOMER-ID.

START CUSTFILE KEY < CUSTOMER-ID.

START CUSTFILE KEY NOT GREATER THAN CUSTOMER-ID.

START CUSTFILE KEY NOT > CUSTOMER-ID.

START CUSTFILE KEY <= CUSTOMER-ID.

START CUSTFILE KEY LESS THAN OR EQUAL TO CUSTOMER-ID.

START CUSTFILE KEY LESS THAN OR EQUAL TO CUSTOMER-ID

   INVALID KEY DISPLAY "INVALID KEY!" LINE 17 COL 10
   NOT INVALID KEY
     DISPLAY "NOT INVALID KEY!" LINE 17 COL 10

END-START.
```

5.3.42 STOP Statement

The STOP statement terminates the current program.

Format 1

```
STOP RUN [ {RETURNING} numeric-1 ]
           {GIVING }
```

Syntax

1. numeric-n is a literal or data item that is numeric.

General Rules

1. A Format 1 STOP RUN statement terminates the program, optionally returning a return code to the operating system. Any OPEN files are CLOSEd.
2. In the Format 1 STOP RUN statement, the RETURNING/GIVING clause returns the value of numeric-1 to the operating system through the special return-code register.
3. RETURNING and GIVING are synonyms.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  STOP-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 NUMERIC-1   PIC 9 VALUE 1.
       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
      *FORMAT 1
      *STOP RUN [ {RETURNING} NUMERIC ]
      *           {GIVING }
           EVALUATE DUMMY
             WHEN "A"
               STOP RUN
             WHEN "B"
               STOP RUN RETURNING NUMERIC-1
             WHEN "C"
               STOP RUN GIVING NUMERIC-1
             WHEN "D"
               STOP RUN RETURNING 1
             WHEN "E"
               STOP RUN GIVING 1
             WHEN OTHER
               CONTINUE
           END-EVALUATE.

           DISPLAY "STOP-1 FINISHED!" LINE 15 COL 10.
           ACCEPT DUMMY LINE 15 COL 30.
           STOP RUN.
```

5.3.43 STRING Statement

The STRING statement concatenates separate literals and data items into a single data string.

General Format

```
STRING { {src-string-1} ...   [DELIMITED BY {delimiter-1}] } ...
                                           {SIZE     }
       INTO target-string-1
       [ WITH POINTER pointer-variable-1   ]
       [ ON OVERFLOW statement-1     ]
       [ NOT ON OVERFLOW statement-2 ]
       [ END-STRING ]
```

Syntax

1. Src-string-n is a alphanumeric data element, literal, or data returned from a function call.
2. pointer-variable-n is a numeric data element with a positive integer value.
3. delimiter-n is a character string.
4. target-string-1 is an alphanumeric data item.
5. statement-n is an imperative statement.

General Rules

1. The STRING statement concatenates consecutive src-string-n, and stores the result in the targe-string data item that follows the INTO phrase.
2. There is no limit to the number of src-string-n identifiers that may be listed.
3. The DELIMITED BY clause provides a way to identify a substring of a src-identifier, for purposes of the STRING operation.
4. The DELIMITED BY delimiter-1 clause causes the STRING operation to stop copying data from the src-string when the delimiter-1 is encountered in the src-string. Delimiter-1 is not copied to the target-string.
5. The DELIMITED BY SIZE clause causes the entire src-string to be copied into the target string.
6. If there is no DELIMITED BY phrase, then DELIMITED BY SIZE is assumed.
7. The WITH POINTER clause provides a way to set a position in the target-string where the STRING statement will copy the selected source using a pointer-variable.
8. The WITH POINTER clause causes an explicit pointer-variable to be referenced by the STRING operation when it begins to copy data from a src-string to a target-string. The value of the pointer-variable is used as the position in the target-string at which to copy the data.
9. When the WITH POINTER clause is used, pointer-variable-n must be set to a positive integer value before the STRING statement is executed. Most commonly, the pointer-variable is initialized to 1, indicating that the STRING statement should begin copying into

the target-string at the first byte of the target-string.

10. The pointer-variable is automatically incremented by 1 for each character that is copied into the target-string.
11. If there is no WITH POINTER clause, the position at which the src-string is copied into the target-string is the next character position after the last character position occupied by the previous src-string.
12. The ON OVERFLOW condition is triggered when the pointer variable does not contain a positive value, or when the length of target-string is less than the combined length of the src-strings named in the STRING statement.
13. When an ON OVERFLOW condition exists, the STRING is terminated, and the ON OVERFLOW statement-1 is executed.
14. The NOT ON OVERFLOW condition exists when the STRING statement has completed, an the ON OVERFLOW condition does not exist.
15. When a NOT ON OVERFLOW condition exists, statement-2 is executed.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. STRING-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ADDRESS-1
         PIC X(40)  VALUE"231/RUE SAINT-HONORE/75001 PARIS/FRANCE".
       01  ADDRESS-2.
         05  ADDRESS-NUM        PIC X(5)       VALUE SPACES.
         05  ADDRESS-NAME       PIC X(20)      VALUE SPACES.
         05  ADDRESS-CITY       PIC X(20)      VALUE SPACES.
         05  ADDRESS-COUNTRY    PIC X(20)      VALUE SPACES.
       01  ADDRESS-3          PIC X(60)        VALUE SPACES.
       01 DUMMY  PIC X.
       PROCEDURE DIVISION.
       MAIN.
           UNSTRING ADDRESS-1 DELIMITED BY "/"
              INTO ADDRESS-NUM
                   ADDRESS-NAME
                   ADDRESS-CITY
                   ADDRESS-COUNTRY.

           DISPLAY "THE STREET #:    " ADDRESS-NUM LINE 4 COL 10.
           DISPLAY "THE STREET NAME: " ADDRESS-NAME LINE 5 COL 10.
           DISPLAY "THE CITY:        " ADDRESS-CITY LINE 6 COL 10.
           DISPLAY "THE COUNTRY:     " ADDRESS-COUNTRY LINE 7 COL 10.

           ACCEPT DUMMY LINE 7 COL 33.
           STRING  ADDRESS-NUM     DELIMITED BY " "
                           " "     DELIMITED BY SIZE
                   ADDRESS-NAME    DELIMITED BY "  "
                           ", "    DELIMITED BY SIZE
                   ADDRESS-CITY    DELIMITED BY "  "
                           ", "    DELIMITED BY SIZE
                   ADDRESS-COUNTRY DELIMITED BY SIZE INTO ADDRESS-3
             ON OVERFLOW
              DISPLAY "STRING FAILED ON OVERFLOW" LINE 10 COL 10
             NOT ON OVERFLOW
              DISPLAY "THE ADDRESS IS " ADDRESS-3 LINE 10 COL 10
           END-STRING.
```

```
           DISPLAY "STRING-1 FINISHED!" LINE 12 COL 10.
           ACCEPT DUMMY LINE 12 COL 30.
           STOP RUN.
```

5.3.44 SUBTRACT Statement

The SUBTRACT Statement performs an arithmetic subtract operation on a number of operands and allows for the storage of the result in a number of data items.

Format 1

```
SUBTRACT {integer-1} ... FROM {integer-data-1 ROUNDED] } ...
         [ ON SIZE ERROR statement-1 ]
         [ NOT ON SIZE ERROR statement-2 ]
         [ END-SUBTRACT ]
```

Format 2

```
SUBTRACT {integer-2} ... FROM integer-3
         GIVING { integer-data-2 [ROUNDED] } ...
         [ ON SIZE ERROR statement-1 ]
         [ NOT ON SIZE ERROR statement-2 ]
         [ END-SUBTRACT ]
```

Format 3

```
SUBTRACT {CORRESPONDING} group-1 FROM group-2 [ROUNDED]
         {CORR }
         [ ON SIZE ERROR statement-1 ]
         [ NOT ON SIZE ERROR statement-2 ]
         [ END-SUBTRACT ]
```

Syntax

1. integer-n is a data element, literal or data returned from a function call that is an integer.
2. integer-data-n is a data item that is an integer.
3. group-n is a group data item containing one or more elementary data items.
4. statement-n is an imperative statement.
5. The SIZE ERROR exception is triggered if the receiving field is not large enough to accommodate the result of the SUBTRACT function.
6. In a SUBTRACT CORRESPONDING operation, elementary items in separate group items must have the same elementary data name for the SUBTRACT function to be performed.

General Rules

1. In a FORMAT 1 SUBTRACT statement with no GIVING clause, all integer-n data items are subtracted, in sequence, from each of the data items following the FROM clause.
2. In a FORMAT 2 SUBTRACT statement containing a GIVING clause, all integer-n data items are subtracted, in sequence, from the data item following the FROM clause with the result stored in the integer-data item following the GIVING clause.
3. The ROUNDED clause is applied when an arithmetic operation produces a result that includes more decimal places than are included in the description of the data item given to hold the final result of the arithmetic operation.
4. For rules regarding the ROUNDED clause, see the entry for ROUNDED in 5.3.1 Common General Rules.
5. The SIZE ERROR exception is triggered if the ON SIZE ERROR clause is present and if the receiving field is not large enough to accommodate the result of the SUBTRACT function.
6. For rules regaring the ON SIZE ERROR clause, see the entry for ON SIZE ERROR in 5.3.1 Common General Rules.
7. In a SUBTRACT CORRESPONDING operation, elementary items in separate group items must have the same elementary data name for the SUBTRACT function to be performed.
8. The SUBTRACT CORRESPONDING operation causes multiple SUBTRACT operations to be performed between elementary data items in separate group items, where the elementary items have the same elementary data name, and are not described as FILLER.
9. The rules for identifying a CORRESPONDING data item for the purposes of the SUBTRACT CORRESPONDING clause are the same as the rules used for the purposes of the MOVE CORRESPONDING clause. For more detail, see the General Rules of the MOVE CORRESPONDING clause.
10. For details on how data items are identified as CORRESPONDING, see 5.3.1 Common General Rules / Rules for identifying CORRESPONDING data elements.
11. In a SUBTRACT CORRESPONDING operation, all SUBTRACT operations will be completed before the ON SIZE ERROR condition will be triggered.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  SUBTRACT-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 FIELD-1 PIC 9V9.
       77 FIELD-2 PIC 9V9.
       77 FIELD-3 PIC9.9.

       01 GROUP-1.
         03 FLD-1    PIC 99 VALUE 5.
         03 FLD-2    PIC 99 VALUE 15.
       01 GROUP-2.
         03 FLD-1    PIC 99 VALUE 10.
         03 FLD-2    PIC 99 VALUE 20.
```

```
       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
      *SUBTRACT {INTEGER-1} ... FROM {INTEGER-2 [ROUNDED] } ...
      *    [ ON SIZE ERROR STATEMENT-1 ]
      *    [ NOT ON SIZE ERROR STATEMENT-2 ]
      *    [ END-SUBTRACT ]
      *
           MOVE 2.5 TO FIELD-1.
           MOVE 3.4 TO FIELD-2.

           SUBTRACT FIELD-1 FROM FIELD-2
             ON SIZE ERROR
               DISPLAY "INVALID SUBTRACTION" LINE 10 COL 10
             NOT ON SIZE ERROR
               DISPLAY FIELD-2 LINE 10 COL 10
           END-SUBTRACT.


      *SUBTRACT {INTEGER-3} ... FROM INTEGER-4
      *    GIVING { INTEGER-5 [ROUNDED] } ...
      *    [ ON SIZE ERROR STATEMENT-1 ]
      *    [ NOT ON SIZE ERROR STATEMENT-2 ]
      *    [ END-SUBTRACT ]
      *

           MOVE 2.5 TO FIELD-1.
           MOVE 3.4 TO FIELD-2.

           SUBTRACT FIELD-1 FROM FIELD-2 GIVING FIELD-3
             ON SIZE ERROR DISPLAY "INVALID SUBTRACTION" LINE 10 COL 10
             NOT ON SIZE ERROR DISPLAY FIELD-3 LINE 10 COL 10
           END-SUBTRACT.


      *SUBTRACT {CORRESPONDING} GROUP-1 FROM GROUP-2 [ROUNDED]
      *         {CORR }
      *    [ ON SIZE ERROR STATEMENT-1 ]
      *    [ NOT ON SIZE ERROR STATEMENT-2 ]
      *    [ END-SUBTRACT ]

           SUBTRACT CORRESPONDING GROUP-1 FROM GROUP-2
             ON SIZE ERROR
               DISPLAY "INVALID SUBTRACTION" LINE 10 COL 10
             NOT ON SIZE ERROR
               DISPLAY FLD-1 OF GROUP-2 LINE 12 COL 10
               DISPLAY FLD-2 OF GROUP-2 LINE 13 COL 10
           END-SUBTRACT.

           DISPLAY "SUBTRACT-1 FINISHED!" LINE 15 COL 10.
           ACCEPT DUMMY LINE 15 COL 30.
           STOP RUN.
```

5.3.45 TRANSFORM Statement

The TRANSFORM statement converts characters in a data item from one character string to another.

General Format

```
TRANSFORM identifier-1 FROM literal-1 TO literal-2
```

Syntax

1. Identifier-n is a data item, literal, or data returned from a function call that is alphanumeric.
2. literal-n is a character string.

General Rules

1. The TRANSFORM statemnt causes a conversion of characters within identifier-1 according to rules established by the ordinal positions of characters in identifier-2 and identifier-3. To clarify with an example, the statement :

   TRANSFORM identifier-1 FROM "AB" TO "CD"

   causes every instance of "A" in identifier-1 (the first character in identifier-2), to be converted to "C" (the first character in identifier-3), and causes every instance of "B" in identifier-1 (the second character in identifier-2) to be replaced by "D" (the second character in identifier-3.

   Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  TRANSFORM-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 IDENTIFIER-1 PIC X(10) VALUE"AAAXXXXBBB".
       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
           TRANSFORM IDENTIFIER-1 FROM "AB" TO "CD".
           DISPLAY IDENTIFIER-1 LINE 10 COL 10.
           DISPLAY "TRANSFORM-1 FINISHED!" LINE 15 COL 10.
           ACCEPT DUMMY LINE 15 COL 30.
           STOP RUN.
```

5.3.46 UNLOCK Statement

The UNLOCK statement removes record locks.

General Format

```
UNLOCK file-1  {RECORD }
               {RECORDS}
```

Syntax

1. file-n is a file described in the File Section with an FD.

General Rules

1. file-1 must be OPEN when the UNLOCK statement is executed.
2. The UNLOCK statement releases record locks for file-1.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. UNLOCK-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT RESWORDS ASSIGN TO "RESWORDS"
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS RESERVED-WORD
           FILE STATUS IS RESWORDS-STAT.

       DATA DIVISION.
       FILE SECTION.
       FD RESWORDS.
       01 RESWORDS-RECORD.
         03 RESERVED-WORD         PIC X(30).
         03 COMMENT               PIC X(20).

       WORKING-STORAGE SECTION.
       77 RESWORDS-STAT PIC XX.
         88 END-OF-RESWORDS  VALUE "10".
       77 DUMMY                   PIC X.

       PROCEDURE DIVISION.
       MAIN.
           OPEN OUTPUT RESWORDS.
           MOVE "ACCEPT" TO RESERVED-WORD.
           MOVE "HELLO WORLD" TO COMMENT.
           WRITE RESWORDS-RECORD.
           CLOSE RESWORDS.

           OPEN I-O RESWORDS.
           MOVE "ACCEPT" TO RESERVED-WORD.
           READ RESWORDS WITH LOCK.

           UNLOCK RESWORDS RECORD.
           DISPLAY "UNLOCK [FILE] RECORD" LINE 5 COL 10.

           MOVE "ACCEPT" TO RESERVED-WORD.
           READ RESWORDS WITH LOCK.

           UNLOCK RESWORDS RECORDS.
           DISPLAY "UNLOCK [FILE] RECORDS" LINE 7 COL 10.

           DISPLAY "UNLOCK-1 FINISHED!" LINE 11 COL 10.
           ACCEPT DUMMY LINE 11 COL 35.

           CLOSE RESWORDS.
           STOP RUN.
```

5.3.47 UNSTRING Statement

The UNSTRING statement separates parts of an existing string that share known delimiters into different data items.

General Format

```
UNSTRING identifier-1
    [ DELIMITED BY [ALL] delimiter-1
    [ OR [ALL] delimiter-2 ] ... ]
    INTO { identifier-2 [ DELIMITER in delimiter-2 ]
    [ COUNT IN integer-data-1 ] } ...
    [ WITH POINTER integer-data-2 ]
    [ TALLYING IN integer-data-3 ]
    [ ON OVERFLOW statement-1 ]
    [ NOTON OVERFLOW statement-2 ]
    [ END-UNSTRING ]
```

Syntax

1. identifier-n is a literal or data item that is not numeric.
2. delimiter-n is an alphanumeric literal.
3. integer-data-n is a numeric data item designed to hold an integer.
4. statement-n is an imperative statement.

General Rules

1. There is no limit to the number of target identifiers that may be listed.
2. The DELIMITED BY clause provides a way to identify a substring of a identifier-1, for purposes of the UNSTRING operation.
3. The DELIMITED BY delimiter-1 clause causes the UNSTRING operation to stop copying data from identifier-1 when delimiter-1 is encountered. Delimiter-1 is not copied to the target-string.
4. The COUNT phrase counts the number of characters in a substring.
5. The WITH POINTER clause provides a way to set a position in the source string from which the UNSTRING statement will copy the selected source using a pointer-variable.
6. The WITH POINTER clause causes an explicit pointer variable to be referenced by the UNSTRING operation when it begins to copy data from a source string to a target string. The value of the pointer-variable is used as the position in the target-string at which to copy the data.
7. When the WITH POINTER clause is used, the pointer variable must be set to a positive integer value before the UNSTRING statement is executed. Most commonly, the pointer variable is initialized to 1, indicating that the UNSTRING statement should begin parsing the source string from the first byte.
8. The pointer-variable is automatically incremented by 1 for each character that is parsed in the source string.

9. If there is no WITH POINTER clause, the position at which the source string is copied into the next target-string is the first character, or, the next character position after the last delimiter character encountered in the original source string.
10. The TALLY phrase counts the number of substrings that have been created by the UNSTRING statement.
11. The ON OVERFLOW condition is triggered when the pointer variable does not contain a positive value, or when the length of target-string is less than the length of the substring identified in the UNSTRING statement.
12. When an ON OVERFLOW condition exists, the UNSTRING is terminated, and the ON OVERFLOW statement-1 is executed.
13. The NOT ON OVERFLOW condition exists when the UNSTRING statement has completed, and the ON OVERFLOW condition does not exist.
14. When a NOT ON OVERFLOW condition exists, statement-2 is executed.
15. The ALL phrase indicates that a sequence of delimiters should be treated as a singled delimiter.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. UNSTRING-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ADDRESS-1
         PIC X(40)  VALUE"231/RUE SAINT-HONORE/75001 PARIS/FRANCE/".
       01  ADDRESS-2.
         05  ADDRESS-NUM        PIC X(5)       VALUE SPACES.
         05  ADDRESS-NAME       PIC X(20)      VALUE SPACES.
         05  ADDRESS-CITY       PIC X(20)      VALUE SPACES.
         05  ADDRESS-COUNTRY    PIC X(20)      VALUE SPACES.
       77 COUNTER-1    PIC 99 VALUE 0.
       77 COUNTER-2    PIC 99 VALUE 0.
       77 COUNTER-3    PIC 99 VALUE 0.
       77 COUNTER-4    PIC 99 VALUE 0.
       77 TALLY-1      PIC 99 VALUE 0.
       01 DUMMY  PIC X.
       PROCEDURE DIVISION.
       MAIN.
           UNSTRING ADDRESS-1 DELIMITED BY "/"
            INTO ADDRESS-NUM, COUNT IN COUNTER-1,
                ADDRESS-NAME, COUNT IN COUNTER-2,
                ADDRESS-CITY, COUNT IN COUNTER-3,
                ADDRESS-COUNTRY, COUNT IN COUNTER-4
                TALLYING IN TALLY-1.

           DISPLAY "THE STREET #:    "  LINE 4 COL 10.
           DISPLAY ADDRESS-NUM LINE 4 COL 28.
           DISPLAY COUNTER-1 LINE 4 COL 50.
      *
           DISPLAY "THE STREET NAME: "  LINE 5 COL 10.
           DISPLAY ADDRESS-NAME LINE 5 COL 28.
           DISPLAY COUNTER-2 LINE 5 COL 50.

           DISPLAY "THE CITY:        "  LINE 6 COL 10.
           DISPLAY ADDRESS-CITY LINE 6 COL 28.
           DISPLAY COUNTER-3 LINE 6 COL 50.
```

```
           DISPLAY "THE COUNTRY:     "  LINE 7 COL 10.
           DISPLAY ADDRESS-COUNTRY LINE 7 COL 28.
           DISPLAY COUNTER-4 LINE 7 COL 50.

           DISPLAY "TALLY:           " LINE 9 COL 10.
           DISPLAY TALLY-1             LINE 9 COL 28.

           DISPLAY "UNSTRING-1 FINISHED!" LINE 12 COL 10.

           ACCEPT DUMMY LINE 12 COL 30.

           STOP RUN.
```

5.3.48 USE Statement

The USE statement determines the error handling that takes place inside the DECLARATIVES section of the PROCEDURE DIVISION.

General Format

```
USE [GLOBAL] AFTER STANDARD {EXCEPTION} PROCEDUREON {{file-1 }... }
                            {ERROR }                { INPUT  }
                                                    { OUTPUT }
                                                    { I-O    }
                                                    { EXTEND }
```

Syntax

1. file is a data file described in the FILE Section.
2. The USE statement may only be used in the DECLARATIVES section of the PROCEDURE DIVISION.
3. file-n is a file described in the File Section with an FD.

General Rules

1. The USE AFTER STANDARD ERROR PROCEDURE clause includes statements to be executed in the event of file errors if programmatic phrases have not been included to handle the file error conditions inline. Such programmatic phrases include the AT END and INVALID KEY phrases. In each of these cases, the statement lists that are executed when the file error conditions are triggered are listed after the AT END/INVALID KEY phrase.
2. AT END conditions are identified with FILE STATUS conditions whose first byte is “1”. INVALID KEY conditions are identified with FILE STATUS conditions whose first byte is “2”. FILE STATUS conditions whose first byte is “0” are considered to be successful I-O operations.
3. The USE AFTER STANDARD ERROR PROCEDURE clause can be associated with file errors in specific files, as represented by the file-1 notation in the General Format, or with specific I-O types, as represented by the INPUT, OUTPUT, I-O, and EXTEND notations in the General Format.
4. If a specific file is named in the USE clause, then the following statement list is executed when an unsuccessful file operation is returned for that file.

5. After the statement list associated with an unsuccessful file operation has been executed, control returns to the statement after the statement that caused the unsuccessful file operation.
6. Unsuccessful file operations are any file operations that return a value to the FILE STATUS variable described in the SELECT statement of the file, where the first byte of the FILE STATUS variable is greater than zero.
7. If a specific file is named in the USE clause, this takes precedence over the more general association with specific I-O types. That is, if there is a USE AFTER STANDARD ERROR PROCEDURE FOR file-1 statement, and a USE AFTER STANDARD ERROR PROCEDURE FOR INPUT statement, and file-1 is OPEN INPUT and returns a file error, the USE AFTER STANDARD ERROR PROCEDURE FOR file-1 clause will take precedence over the USE AFTER STANDARD ERROR PROCEDURE ON INPUT clause.
8. The USE GLOBAL phrase allows the error handling described by the USE statement to be applicable in all of the programs in the entire compilation unit.
9. EXCEPTION and ERROR are synonyms.
10. The statements that can generate unsuccessful file operations are CLOSE, DELETE, OPEN, READ, REWRITE, START, UNLOCK, and WRITE.

Code Sample:

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  USE-1
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
           SELECT CUSTFILE ASSIGN TO "CUSTOMER"
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS CUSTOMER-ID
           FILE STATUS IS CUSTOMER-STAT.
       DATA DIVISION.
       FILE SECTION.
       FD CUSTFILE.
       01 CUSTOMER-RECORD.
         03 CUSTOMER-ID     PIC 9(5).
         03 CUSTOMER-NAME   PIC X(25).
         03 CUSTOMER-ADDR   PIC X(25).
         03 CUSTOMER-CITY   PIC X(25).
         03 CUSTOMER-STATE  PIC XX.
         03 CUSTOMER-PHONE  PIC X(10).

       WORKING-STORAGE SECTION.
       77 CUSTOMER-STAT PIC XX.
       77 DUMMY             PIC X.

       PROCEDURE DIVISION.
       DECLARATIVES.
       USE-1-ERR-HANDLING SECTION.
           USE AFTER STANDARD ERROR PROCEDURE ON CUSTFILE.
       CUSTFILE-ERR.
           DISPLAY "USE AFTER STANDARD ERROR ON [FILENAME]"
                   LINE 8 COL 10.

           DISPLAY "FILE STATUS: " LINE 10 COL 10.
           DISPLAY CUSTOMER-STAT LINE 10 COL 24.
```

```
           ACCEPT DUMMY LINE 10 COL 30.
           DISPLAY "USE-1 FINISHED!" LINE 12 COL 10.
           STOP RUN.
       END DECLARATIVES.
       MAIN.
           OPEN OUTPUT CUSTFILE.
           MOVE 11111 TO CUSTOMER-ID.
           MOVE "JOHN SMITH" TO CUSTOMER-NAME.
           MOVE "101 MAIN ST" TO CUSTOMER-ADDR.
           MOVE "SAN DIEGO" TO CUSTOMER-CITY.
           MOVE "CA" TO CUSTOMER-STATE.
           MOVE "6195551212" TO CUSTOMER-PHONE.
           WRITE CUSTOMER-RECORD.

           MOVE 11111 TO CUSTOMER-ID.
           MOVE "JOHN SMITH" TO CUSTOMER-NAME.
           MOVE "101 MAIN ST" TO CUSTOMER-ADDR.
           MOVE "SAN DIEGO" TO CUSTOMER-CITY.
           MOVE "CA" TO CUSTOMER-STATE.
           MOVE "6195551212" TO CUSTOMER-PHONE.
           WRITE CUSTOMER-RECORD.

           STOP RUN.
```

5.3.49 WRITE Statement

The WRITE statement adds a record to a data file.

Format 1

The Format 1 WRITE statement writes a record to a sequential file, and includes language for writing to a PRINT file, which is a special form of a line sequential file. The

```
WRITE record-name [ FROM identifier-1 ]
     [ {BEFORE} ADVANCING { integer-1 [LINE           ] } ]
       {AFTER }             [LINES          ]
                            [PAGE           ]
                            [ mnemonic-name ]
     [ AT {END-OF-PAGE} statement-1 ]
        {EOP }
     [ NOT AT {END-OF-PAGE} statement-2 ]
           {EOP }
     [ END-WRITE ]
```

Syntax

1. identifier-n is a data element, literal, or data returned from a function call.
2. integer-n is a data element, literal or data returned from a function call that is an integer.
3. statement-n is an imperative statement.

General Rules

1. The file in which record-name is described in the FILE SECTION must be OPEN when the WRITE statement executes.
2. Record-name is an 01-level record name defined in a File Description (FD).
3. The FROM phrase causes the data in identifier-1to be copied to record-name before the execution of the WRITE statement. Identifier-1 may be in the form of a FUNCTION call.:
4. The ADVANCING phrase should only be used with files described with ORGANIZATION IS LINE SEQUENTIAL.
5. The BEFORE ADVANCING integer-1 LINES phrase causes record-name to be written BEFORE introducing line-feeds to the output record. The number of line-feeds is described with integer-1.
6. The AFTER ADVANCING integer-1 LINES phrase causes record-name to be written AFTER introducing line-feeds to the output record. The number of line-feeds is described with integer-1.
7. The BEFORE ADVANCING integer-1 PAGES phrase causes record-name to be written BEFORE introducing page-feeds to the output record. The number of page-feeds is described with integer-1.
8. The AFTER ADVANCING integer-1 PAGES phrase causes record-name to be written AFTER introducing page-feeds to the output record. The number of page-feeds is described with integer-1.
9. Files that contain a LINAGE clause automatically causes lines feeds to be written to the file as stipulated in the LINES AT BOTTOM and LINES AT TOP clauses.
10. Files that contain a LINAGE clause and a FOOTING clause trigger the END-OF-PAGE condition if a WRITE statement causes any data to be written into the FOOTING area, as described in the FD.
11. If the END-OF-PAGE condition is trapped by an END-OF-PAGE phrase, the associated statement-list is executed, and then control is passed to the next statement after the WRITE statement.
12. The NOT AT END-OF-PAGE condition exists after a WRITE in files that contain a LINAGE clause, and where the WRITE does not cause the internal counter maintained by the LINAGE clause to be reached. If the NOT AT END-OF-PAGE condition is trapped by a NOT AT END-OF-PAGE phrase, the associated statement-list is executed, and then control is passed to the next statement after the WRITE statement.
13. A successful WRITE statement to a sequential file causes the data in record-name to be written to the end of the file.
14. WRITEing a LINE SEQUENTIAL record to a named pipe is supported.
15. The WRITE statement updates the FILE STATUS variable.

Format 2

The Format 2 WRITE statement writes a record to an indexed file, or a relative file, and includes language for trapping the INVALID KEY condition.

```
WRITE record-name [ FROM identifier-1 ]
       [ WITH [NO] LOCK ]
       [ INVALID KEY statement-1 ]
```

```
        [ NOT INVALID KEY statement-2 ]
        [ END-WRITE ]
```

Syntax

1. identifier-n is a data element, literal, or data returned from a function call.
2. statement-n is an imperative statement.

General Rules

1. The file in which record-name is described in the FILE SECTION must be OPEN when the WRITE statement executes.
2. Record-name is an 01-level record name defined in a File Description (FD).
3. The FROM phrase causes the data in identifier-1to be copied to record-name before the execution of the WRITE statement.
4. The WITH LOCK phrase causes the record to be LOCK'ed for the duration of the WRITE statement.
5. The WITH NO LOCK phrase indicates that the record is not LOCK'ed for the duration of the WRITE statement.
6. The INVALID KEY condition exists if a file-status error is generated by the WRITE statement. If the INVALID KEY condition is trapped by an ON INVALID KEY phrase, the associated statement-list is executed, and then control is passed to the next statement after the WRITE statement.
7. Any of the following circumstances triggers the INVALID KEY condition:
    a. A WRITE to a relative file is attempted, and a record with the same RELATIVE KEY already exists.
    b. A WRITE to an indexed file is attempted, and a record with the same RECORD KEY already exists.
    c. A WRITE to an indexed file is attempted, and a record with the same ALTERNATE RECORD KEY already exists, and the ALTERNATE RECORD KEY does not allow duplicates.
8. If the INVALID KEY condition is not trapped by an ON INVALID KEY phrase, it can be trapped in DECLARATIVES with the appropriate USE statement. If it is not trapped in DECLARATIVES, it causes the program to abort.
9. The NOT INVALID KEY condition exists after a WRITE statement is executed successfully. If the NOT INVALID KEY condition is trapped by a NOT ON INVALID KEY phrase, the associated statement-list is executed, and then control is passed to the next statement after the WRITE statement.
10. A successful WRITE statement to a relative file causes the data in record-name to be written to the position in the relative file described by the data item containing the relative key.
11. A successful WRITE statement to an indexed file causes the index, and data portions of the file to be updated.
12. The WRITE statement updates the FILE STATUS variable.

Code Sample

```
...
       SELECT PRINT-FILE-1 ASSIGN TO "PRINTER".

       SELECT IDX-FILE-1 ASSIGN TO "IDX-FILE-1"
       ORGANIZATION IS INDEXED
       ACCESS IS DYNAMIC
       RECORD KEY IS IDX-KEY
       FILE STATUS IS FILE-1-STAT.
...
   FD PRINT-FILE-1.
   01 PRINT-RECORD       PIC X(60).

   FDFILE-1.
   01 FILE-1-RECORD.
     03 FILE-KEY PIC X(10).
...
   77 FILE-1-STAT PIC XX.
   01 WS-IDX-FILE PIC X(10).
...

       WRITE PRINT-RECORD BEFORE ADVANCING 1 LINE.

       WRITE PRINT-RECORD AFTER ADVANCING 1 LINE.

       WRITE PRINT-RECORD AFTER ADVANCING 2 LINES
         AT END-OF-PAGE CONTINUE
         NOT AT END-OF-PAGE CONTINUE
       END-WRITE.

       WRITE PRINT-RECORD AFTER ADVANCING PAGE.

       WRITE PRINT-RECORD BEFORE ADVANCING PAGE.

       WRITE FILE-1-RECORD.

       WRITE FILE-1-RECORD
         INVALID KEY
           DISPLAY "INVALID KEY!" LINE 10 COL 10
         NOT INVALID KEY
           CONTINUE
       END-WRITE.

       WRITE FILE-1-RECORD FROM WS-IDX-FILE.
```

5.3.50 XML Generate Statement

The XML GENERATE statement converts the data items in identifier-2 into an XML document that is stored in identifier-1.

A converted data element stores the value of the data item, the name of the data element, and XML markup. The element names are derived from the data names in identifier-2.

General Format

```
XML GENERATE identifier-1 FROM identifier-2 [COUNT [IN] identifier-3]
  [[ON] EXCEPTION imperative-statement-1]
  [NOT [ON] EXCEPTION imperative-statement-2]
  [END-XML]
```

Syntax Rules

1. identifier-1 is the data item into which the XML document is generated. identifier-1 must be declared as an elementary or group item of category alphanumeric. identifier-1 must not overlap identifier-2 or identifier-3.
2. identifier-1 must be large enough to hold the XML document. A commonly used guideline is to give identifier-1 a size that is 5-10 times the size of identifier-2. If identifier-1 is too small, the XML GENERATE statement will produce an error condition.
3. identifier-2 is the group or elementary item to be converted into an XML document. It must not overlap identifier-1 or identifier-3.
4. identifier-3 is a numeric data item. It must not overlap identifier-1 or identifier-2.

General Rules

1. identifier-1 must not be described with the JUSTIFIED clause, and cannot be a function identifier.
2. If identifier-1 is larger than the XML document, the trailing bytes will not be altered by the XML Generate operation. Data that was present from a previous GENERATE statement could still be present, and not overwritten. For this reason, it is good practice to INITIALIZE identifier-1 before performing the XML GENERATE statement. Alternatively, you could use identifier-3 as a reference modifier when referring to the data in identifier-1.
3. identifier-2 cannot be reference-modified, and cannot be described with the RENAMES clause.
4. XML GENERATE will ignore the following in identifier-2:
    a. unnamed elementary data items, or FILLER data items
    b. slack bytes inserted for SYNCRONIZED items
    c. any items described with or subordinate to a REDEFINES clause.
    d. any items decribed with the RENAMES clause,
    e. any group item whose subordinate data items are all ignored.
5. There must be at least one unignored elementary data item in the identifier-2 group item. \
6. Excluding ignored data items, elementary data items in the identifier-2 group item must be CLASS ALPHABETIC, ALPHANUMERIC, NUMERIC, or be an INDEX item.
7. The COUNT IN phrase manages the count of generated XML characters (in bytes) that are stored in identifier-3.
8. identifier-3 must be an integer data item without the symbol "P" in its picture string.

ON EXCEPTION phrase

1. An EXCEPTION condition exists when an error occurs during the generation of the XML

document. As an example, if identifier-1 is not large enough to hold the XML document, an EXCEPTION condition will be triggered. When an EXCEPTION condition is triggered, control passes to imperative-statement-1. The contents of identifier-1 are undefined. If a COUNT IN phrase was used, the number of characters generated can be retrieved in identifier-3.

NOT ON EXCEPTION phrase

1. If the ON EXCEPTION phrase is specified, and no EXCEPTION condition is triggered, then control passes to imperative-statement-2.

END-XML phrase

1. The END-XML phrase delimits the scope of both XML GENERATE and XML PARSE statements.

Nested XML statements

1. An XML Generate or XML Parse statement located in an ON EXCEPTION imperative-statement is referred to as a nested (or conditional) XML statement. The scope of a conditional XML GENERATE or XML PARSE statement is terminated by either an END-XM phrase at the same level of nesting, or by a separator period.

Special Registers
XML-CODE

1. The XML-CODE special register indicates whether an XML GENERATE statement executed successfully or an exception occurred during XML generation. A successful execution of the XML GENERATE statement causes the XML-CODE special register to be set to 0. Exception conditions return non-zero error codes to the XML-CODE special register.
2. The XML-CODE special register is implicitly defineds as:
   ```
   XML-CODE PICTURE S9(9) USAGE BINARY VALUE 0.
   ```
   Details on handling non-zero error codes is detailed in the IBM Enterprise COBOL Programming Guide, in the "Handling XML Parse exceptions" chapter.

Converting data values, and element names
COBOL-IT uses IBM rules for converting elementary data items to character format, for data trimming, and for deriving element names.

Converting elementary data items is the process of translating different data types into a character format. Alphanumeric data values are unchanged. Rules for converting elementary data items are detailed in the IBM Enterprise COBOL Programming Guide.

Data trimming is the application of rules that determine how leading and trailing zeroes, and spaces are handled for a variety of different cases. Rules for applying data trimming are detailed in

the IBM Enterprise COBOL Programming Guide.

Element naming is the process of translating element data names in identifier-2 into element tag names in the XML document. In most cases, you will see that your element tag name corresponds exactly to yur element name. However, there are cases where data names may contain characters that are illegal in XML. Rules for element naming are detailed in the IBM Enterprise COBOL Programming Guide.

5.3.51 XML PARSE Statement

The XML PARSE statement parses an XML document and returns data values to their corresponding data elements. This data can then be processed by the COBOL program.

General Format

```
XML PARSE identifier-1
PROCESSING PROCEDURE [IS]
        procedure-name-1 [THROUGH procedure-name-2]
                          THRU
  [[ON] EXCEPTION imperative-statement-1]
  [NOT [ON] EXCEPTION imperative-statement-2]
  [END-XML]
```

Syntax Rules

1. identifier-1 is an alphanumeric data item that holds an XML document.
2. The PROCESSING PROCEDURE phrase specifies the name of a procedure to respond to the events generated by the XML PARSE statement.
3. procedure-name-1 and procedure-name-2 define a consecutive sequence of operations to execute, beginning at the procedure named by procedure-name-1 and ending with the execution of the procedure named by procedure-name-2.
4. procedure-name-1, procedure-name-2 indicates a section or paragraph in the PROCEDURE DIVISION. Procedure-name-1 and procedure-name-2 may not be in a declarative section.

General Rules

1. Control passes between the parser and procedure-name1. The parser generates an XML event, procedure-name-1 (or the code executed in procedure-name-1 through procedure-name 2) handles the event, and control is then returned to the parser.
2. The processing procedure interprets the information returned from the parser, and handles it programmatically. When the processing procedure (or procedures) is complete control is returned to the XML parser.
3. The processing procedure can terminate the run unit with a STOP RUN statement.

ON EXCEPTION phrase

1. An EXCEPTION condition exists when an error occurs during the XML PARSE operation. The parser signals the exception by passing control to the processing procedure and updating XML-EVENT and XML-CODE special registers. If the ON EXCEPTION phrase is specified, control passes to imperative-statement-1. For details on the XML-EVENT and XML-CODE special registers, see your IBM documentation.

NOT ON EXCEPTION phrase

1. If the ON EXCEPTION phrase is specified, and no EXCEPTION condition is triggered, then control passes to imperative-statement-2.

END-XML phrase

1. The END-XML phrase delimits the scope of both XML GENERATE and XML PARSE statements.

Nested XML statements

1. An XML Generate or XML Parse statement located in an ON EXCEPTION imperative-statement is referred to as a nested (or conditional) XML statement. The scope of a conditional XML GENERATE or XML PARSE statement is terminated by either an END-XML phrase at the same level of nesting, or by a separator period.

Special Registers

1. In the control flow between the parser and the processing procedure, the parser returns control to the processing procedure, and updates the special registers XML-CODE, XML-EVENT, and XML-TEXT.

XML-CODE

1. The XML-CODE special register is implicitly defineds as:

   01 XML-CODE PICTURE S9(9) USAGE BINARY VALUE 0.

2. The XML-CODE special register communicates status between the XML parser and the processing procedure defined in an XML PARSE statement. The XML-CODE special register also indicates whether an XML GENERATE statement executed successfully or an exception occurred during XML generation.
3. When an XML PARSE statement terminates, the special register XML-CODE is populated either with a "0" to indicate a successful operation, or with a non-zero error code.
4. Details on handling non-zero error codes is detailed in the IBM Enterprise COBOL Programming Guide, in the "Handling XML Parse exceptions" chapter.

XML-EVENT

1. The XML_EVENT special register is implicitly defined as:

   01 XML-EVENT USAGE DISPLAY PICTURE X(30) VALUE SPACE.

2. The XML-EVENT special register communicates event information from the XML parser to the processing procedure defined in an XML PARSE statement.
3. The XML-EVENT special register is set to the name of the XML event.
4. XML events and associated special register contents are listed in the IBM COBOL Enterprise COBOL Programming Guide.

XML-TEXT

1. XML-TEXT is an elementary alphanumeric data item of the length of the contained XML document fragment. The length of XML-TEXT can vary from 0 to 16,777,215 bytes. There is no equivalent COBOL data description entry.
2. The XML-TEXT special register contains document fragments that are of class alphanumeric.
3. The XML parser sets XML-TEXT to the document fragment associated with an XML-EVENT before transferring control to the processing procedure when the operand of the XML PARSE statement is an alphanumeric data item. Use the LENGTH function for XML-TEXT to determine the number of bytes that XML-TEXT contains.
4. XML-TEXT cannot be used as a receiving item.
5. XML events and associated XML-TEXT contents are listed in the IBM COBOL Enterprise COBOL Programming Guide.

www.cobol-it.com

June, 2018