

# **COBOL 85 Language Reference**

**( COBOL ReSource )**

**or**

**( Wang VS )**

Document Version 1.00  
1 September 2000

SRDI does not warrant that the Software or documentation will meet your requirements, that operation of the Software will be uninterrupted or error-free, or that all Software errors will be corrected. SRDI will not guarantee the accuracy of the documentation. All Reasonable steps will be taken to correct documentation when possible. SRDI is not responsible for problems caused by changes in the operating characteristics of computer hardware or computer operating systems which are made after the release of the Software nor for problems in the interaction of the Software with other software. SRDI will have no responsibility to replace or refund the license fee of media damaged by accident, abuse or misapplication. **SRDI issues a license to use the software**, this does not imply any right to ownership and is **not transferable**.

THE ABOVE WARRANTIES ARE EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES, WHETHER EXPRESSED OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY SRDI, ITS EMPLOYEES, DISTRIBUTORS, DEALERS OR AGENTS WILL INCREASE THE SCOPE OF THE ABOVE WARRANTIES OR CREATE ANY NEW WARRANTIES. SHOULD THE SOFTWARE PROVE DEFECTIVE, YOU ASSUME THE RISK OF PAYING THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION AND ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES. IN NO EVENT WILL SRDI BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING WITHOUT LIMITATION DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION AND THE LIKE) ARISING OUT OF THE USE OR THE INABILITY TO USE THIS PRODUCT EVEN IF SRDI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**This document may be copied in full for use by licensed users of Cobol Resource or Wang VS. This notice must be present along with all other parts of this document. The document must remain in the same form as distributed by SRDI. If the file is an Adobe pdf file it must remain so.**

This document may not be duplicated in part. No portion of this document may be distributed or used in any form without the prior written consent of SRDI.

**© Copyright 2000 SRDI. - ALL RIGHTS RESERVED.**

# Contents

How to Use This Guide .....	9
What You Need to Know .....	9
Organization .....	9
Syntax Conventions .....	10
Related Documents .....	11
Chapter 1 COBOL 85 Overview .....	13
Introduction.....	13
New Features .....	13
Scope Terminators .....	14
Nested Programs .....	14
External Data .....	14
PERFORM Statement Enhancements .....	15
CONTINUE Statement .....	15
EVALUATE Statement .....	15
CALL Statement.....	15
Reference Modification .....	16
Intrinsic Functions .....	16
Symbolic Characters.....	16
Miscellaneous Changes .....	16
ANSI Conformance .....	17
Wang Extensions .....	17
File Sharing Support .....	18
Database Support .....	18
Interactive Processing.....	18
Program Development Tools.....	19
Chapter 2 Program Structure .....	20
Introduction.....	20
Division Components .....	20
Nested Source Programs.....	21
Language Structure .....	24
Character Set.....	24
The COBOL Character Set.....	24
Character-Strings and Separators .....	25
Words.....	26
Literals .....	29
Reference Format.....	29
Data Description .....	32
Logical and Physical Records.....	33
Buffering.....	33
Data Structure .....	33
Classes of Data .....	35
Representation .....	36
Character Representation.....	36
Algebraic Signs.....	36
Standard Alignment Rules.....	36

Scope of Names and Uniqueness of Reference .....	37
Qualification .....	37
Subscripting and Indexing .....	39
Condition-Names .....	40
Corresponding Data Items .....	41
Reference Modification .....	41
Chapter 3 Identification Division .....	43
Introduction.....	43
PROGRAM-ID Paragraph.....	43
Initial State of a Program.....	44
Common Programs .....	44
Additional Paragraphs .....	44
Chapter 4 Environment Division.....	45
Introduction.....	45
Configuration Section.....	45
SOURCE-COMPUTER Paragraph .....	46
OBJECT-COMPUTER Paragraph .....	47
SPECIAL-NAMES Paragraph.....	48
FIGURATIVE-CONSTANTS Paragraph .....	51
Input-Output Section .....	52
FILE-CONTROL Paragraph .....	52
FILE-CONTROL Clauses .....	57
I-O-CONTROL Paragraph .....	62
Chapter 5 Data Division .....	63
Introduction.....	63
Data Description Entry .....	64
Data Description Clauses.....	66
BLANK WHEN ZERO Clause .....	66
Data-Name-FILLER Clause .....	66
EXTERNAL Clause .....	66
GLOBAL Clause .....	67
JUSTIFIED Clause .....	67
Level-Number Clause.....	67
OCCURS Clause .....	68
PICTURE Clause.....	69
REDEFINES Clause.....	75
RENAMES Clause .....	76
SIGN Cause .....	76
SYNCHRONIZED Clause .....	77
USAGE Clause .....	78
VALUE Clause.....	80
Workstation Screen Description Entry .....	82
COLUMN Clause .....	83
ROW Clause .....	84
RANGE Clause.....	84
SOURCE and VALUE Clauses.....	85
OBJECT Clause.....	85
OCCURS Clause .....	86
Noncontiguous Data .....	86
File Section .....	87

File Description Entry.....	87
File Description Clauses .....	90
BLOCK CONTAINS Clause.....	90
CODE-SET Clause .....	90
DATA RECORDS Clause.....	91
EXTERNAL Clause .....	91
GLOBAL Clause .....	92
LABEL RECORDS Clause .....	92
RECORD Clause .....	92
VALUE Clause.....	95
LINEAGE Clause.....	97
SORT-MERGE File Description Entry .....	100
Working-Storage Section.....	100
Initial Values.....	101
Linkage Section .....	101
External Objects and Internal Objects .....	101
Local Names and Global Names .....	102
Scope of Names .....	102
Chapter 6 Procedure Division.....	103
Introduction.....	103
Statements and Sentences .....	103
Organization .....	104
Arithmetic .....	106
Arithmetic Operators .....	106
Arithmetic Expressions.....	107
Arithmetic Expression Evaluation Rules.....	107
Composite of Operands .....	107
Multiple Receiving Items in Arithmetic Statements .....	108
Conditions.....	108
Relation Conditions .....	109
Class Conditions .....	110
Condition-Name Tests .....	111
Sign Condition .....	112
Modified Data Tag Conditions .....	112
Mask Tests .....	112
Simple and Complex Conditions .....	113
Order of Condition Evaluation .....	115
Exception Handling Phrases .....	115
Declaratives Execution .....	116
Program Termination.....	116
Other Considerations .....	116
Chapter 7 Procedure Division Statements .....	118
Introduction.....	118
ACCEPT Statement.....	119
ADD Statement.....	121
ALTER Statement .....	123
BEGIN Statement .....	124
CALL Statement.....	125
CANCEL Statement .....	128
CLOSE Statement.....	129

Explanation of CLOSE Statement Formats .....	131
Explanation of CLOSE Statements for File Categories .....	133
COMMIT Statement .....	134
COMPUTE Statement .....	135
CONTINUE Statement .....	136
DELETE Statement .....	137
DISPLAY Statement .....	139
DISPLAY AND READ Statement .....	140
DIVIDE Statement .....	142
ENTER Statement .....	145
EVALUATE Statement .....	146
EXIT Statement .....	149
EXIT PROGRAM Statement .....	149
FREE ALL Statement .....	150
GO TO Statement .....	150
HOLD Statement .....	151
IF Statement .....	153
INITIALIZE Statement .....	154
INSPECT Statement .....	156
MERGE Statement .....	161
MOVE Statement .....	165
1. MULTIPLY Statement .....	170
OPEN Statement .....	172
PERFORM Statement .....	176
READ Statement .....	180
RELEASE Statement .....	185
RETURN Statement .....	186
REWRITE Statement for Sequential Files .....	188
REWRITE Statement for Relative and Indexed Files .....	190
ROLLBACK Statement .....	193
SET Statement .....	198
SORT Statement .....	202
START Statement for Workstation Files .....	207
START Statement for Relative and Indexed Files .....	208
STOP Statement .....	211
STRING Statement .....	212
SUBTRACT Statement .....	214
UNSTRING Statement .....	216
USE Statement .....	219
WRITE Statement for Sequential Files .....	222
WRITE Statement for Relative Files .....	225
WRITE Statement for Relative and Indexed Files .....	227
Chapter 8 Intrinsic Functions .....	230
Introduction .....	230
Language Concepts .....	230
General Description .....	231
Arguments .....	231
Types of Functions .....	232
Returned Value .....	233
Date Conversion .....	233

Function Definitions .....	233
ACOS.....	233
ANNUITY .....	234
ASIN .....	235
ATAN .....	235
CHAR .....	236
COS.....	236
CURRENT-DATE.....	237
DATE-OF-INTEGERS.....	238
DAY-OF-INTEGERS.....	238
FACTORIAL .....	239
INTEGER .....	239
INTEGER-OF-DATE.....	240
INTEGER-OF-DAY .....	240
INTEGER-PART.....	241
LENGTH .....	242
LOG .....	243
LOG10 .....	243
LOWER-CASE.....	244
MAX .....	245
MEAN.....	246
MEDIAN .....	246
MIDRANGE.....	247
MIN.....	247
MOD .....	248
NUMVAL.....	249
NUMVAL-C .....	250
ORD .....	251
ORD-MAX .....	251
ORD-MIN.....	252
PRESENT-VALUE .....	252
RANDOM.....	253
RANGE.....	254
REM.....	254
REVERSE.....	255
SIN .....	255
SQRT .....	256
STANDARD-DEVIATION .....	256
SUM.....	257
TAN .....	257
UPPER-CASE .....	258
VARIANCE.....	258
WHEN-COMPILED.....	259
Appendix A Reserved Words .....	260
Appendix B Compiler Options and Directives .....	264
Appendix C Field Attribute Characters .....	286
Appendix D.....	291
Workstation Screen Order Area.....	291
Appendix E File Status Key Values .....	296
Appendix F Printer Control Characters .....	311

Appendix G Intermediate Results.....	313
Appendix H VS Extension Rights .....	315
Appendix I Wang COBOL 85, ANSI, and FIPS Standards .....	320
Appendix J Obsolete Elements in COBOL 85 .....	327
Appendix K Link Editing and the COBOL 85 Runtime .....	329



# How to Use This Guide

This guide explains and details the use of COBOL 85 (Common Business Oriented Language, 1985 ANSI Standard) for use on both the UNIX® platform and the Wang VS system.

## What You Need to Know

This guide is written for experienced programmers and is intended as a reference. Readers should be familiar with developing applications in COBOL or C language with either the Wang VS system or a UNIX platform.

## Organization

This guide is organized as follows:

- Chapter 1** - Introduces the COBOL 85 language and includes an overview of new features and Wang extensions.
- Chapter 2** - Explains the COBOL 85 program structure and language rules.
- Chapter 3** - Presents the language elements of the Identification Division.
- Chapter 4** - Presents the Environment Division language elements.
- Chapter 5** - Presents the Data Division language elements.
- Chapter 6** - Discusses the Procedure Division arithmetic, conditions, and exception handling.
- Chapter 7** - Presents the Procedure Division statements.
- Chapter 8** - Presents the Intrinsic Functions.

The guide also contains several appendixes:

- Appendix A** - Presents the list of reserved words in standard COBOL and in the Wang implementation of COBOL.
- Appendix B** - Explains the Compiler options and source text directives.
- Appendix C** - Discusses workstation fields and how field attribute characters control the operation of the workstation.
- Appendix D** - Describes the use of the workstation screen order area for controlling screen and workstation actions.
- Appendix E** - Lists and explains file status return codes.
- Appendix F** - Explains the hexadecimal characters used to control the writing of a record to a printer file.
- Appendix G** - Explains the storing of intermediate results of arithmetic operations.
- Appendix H** - Describes the use of VS extension rights.
- Appendix I** - Compares VS, ANSI, and FIPS COBOL 85 standards.
- Appendix J** - Specifies elements from previous versions of COBOL that are obsolete in COBOL 85.
- Appendix K** - Describes link editing and COBOL 85 runtime.

## Syntax Conventions

This guide makes extensive use of formats, syntax rules, and general rules:

- Formats are generalized models of the components of an entry or statement.
- Syntax rules govern the *arrangement* of those components.
- General rules govern the *use* of the components within a program.

This guide uses the conventional system of notation to represent the COBOL 85 language, as shown in the following list:

- Format diagrams are presented in a mono space (fixed-width) font:

ACCEPT data-name-1

UPPERCASE is used for keywords and optional words. Mandatory keywords are underlined. They are not mandatory when they lie within an optional part of the format or when they are enclosed in brackets [ ]. Optional words are never underlined.

Lowercase is used when a user-defined word must be supplied. Rules accompanying the format indicate the conventions for these words.

Some lowercase words have a numeric suffix, for example, data-name-3, to make the wording of the rules more intelligible. The suffix does not change the syntactical definition of the word. The numeric suffix changes the word from a common noun to a proper one.

The special character words, plus sign (+), minus sign (-), greater than (>), less than (<), and equal sign (=), are always mandatory, even though they are not underlined.

Square brackets [ ] indicate that the enclosed item is optional. When the brackets enclose a vertical list of two or more items, one or none of the items can be used

data-name-1  
ROLLBACK  
literal-1

Curly Braces { } indicate that one of the enclosed items must be chosen.

file-name-1  
COPY  
literal-1

Ellipsis (...) indicates that the associated material can be repeated. (If there is a limit on the number of times this repetition can occur, it is specified in the rules.) If material in brackets

or braces is to be repeated, the entire unit must be repeated. The ellipsis symbol is associated with a lowercase word, a set of brackets, or a set of braces.

*Note: Brackets, braces, choice indicators, and ellipses are not part of the COBOL 85 character set.*

Some of the formats contain a word or words followed by the word *clause*, *statement*, or *entry*, indicating a clause, statement or entry that is described in another format.

## Related Documents

UNIX users may want to refer to other documents in the COBOL ReSource document set (part numbers are indicated in parentheses for ordering purposes):

- *COBOL ReSource Programmer's Guide*  
An introduction to the Wang implementation of the ANSI COBOL 85 standard for UNIX platforms.
- *COBOL ReSource Debugger Reference*  
A reference guide to the COBOL ReSource Debugger for locating and correcting errors in COBOL and C language programs.
- *COBOL ReSource Toolkit Reference, Volume I: Utilities Reference*  
A reference guide describing the utilities available with the COBOL ReSource development package.
- *COBOL ReSource Toolkit Reference, Volume II: Subroutine Reference*  
A reference guide to the user subroutines available with the COBOL ReSource product.
- *COBOL ReSource User's Guide (715-4948)*  
Information about configuring, installing, and administering the COBOL ReSource product, as well as instructions for migrating VS COBOL applications to a UNIX platform.

VS users may want to refer to other documents in the VS document set (part numbers are indicated in parentheses for ordering purposes):

- *COBOL 85 Migration Guide*  
Instructions for upgrading from COBOL 74 to COBOL 85.
- VS Programmer's Introduction  
An introduction to programming on the Wang VS system.
- VS Programmer's Guide to COBOL  
An introduction to VS COBOL for both novices and experienced programmers.
- VS Procedure Language Reference  
A reference guide to the procedures used to run system functions and to supply parameter information.
- VS Program Development Tools  
A guide to the EDITOR, LINKER, and SYMBOLIC DEBUGGER programs.
- VS COBOL Quick Reference  
A complete list of COBOL language formats.
- VS Extended Data Management System (XDMS) Reference  
A guide to indexed-plus file organization.
- VS System Operator's Guide  
A reference guide for system administrators and console operators on the functions and procedures that initialize, manage, and control the VS systems.
- VS Operating System Services  
A reference guide for programmers on how to use system routines to control the execution and interaction of programs.

# Chapter 1 COBOL 85 Overview

## Introduction

COBOL (Common Business Oriented Language) is the most widely used programming language in the world. Its popularity stems from its self documenting English-like statements, its ability to handle large volumes of data efficiently, its extensive file processing capabilities, and its ability to produce large reports.

COBOL statements are made up of verbs, connectives, and conventional arithmetic symbols; and, like English, COBOL statements are organized as paragraphs, sentences, and clauses. Unlike English, however, COBOL rules are stringent.

This manual presents the elements of COBOL, their functions, and the rules governing their use. It is intended as a reference manual for experienced COBOL programmers; it is not an introductory text.

## New Features

The American National Standards Institute (ANSI) specifications for Programming Language COBOL (ANSI X3.23-1974) are referenced in this manual as COBOL 74. COBOL 85 contains many features that were not present in COBOL 74. Most of the changes were made to make it easier to create structured programs. The following sections briefly introduce the major changes. Appendix I presents a complete list of the differences between the COBOL 74 and COBOL 85 standards.

## Scope Terminators

Scope Terminators delimit the scope of statements. This is probably the most important innovation for COBOL 85. Under COBOL 74, a conditional statement within another conditional statement cannot be terminated without terminating the entire construct. In COBOL 85, you can terminate a single statement within a construct without terminating the rest of the statements. Scope Terminators take the form END-verb. The following example shows the use of a Scope Terminator with the IF statement:

```
IF condition-1 IS TRUE
  THEN
    IF condition-2 IS TRUE
      THEN PERFORM DO-RTN-1
      ELSE PERFORM DO-RTN-2
    END-IF
    PERFORM DO-RTN-3
  ELSE
    PERFORM DO-RTN-4
END-IF.
```

In this example, paragraph DO-RTN-3 is performed whenever condition-1 is true, without regard for the truth value of condition-2. (The state of condition-2 determines whether DO-RTN-1 or DO-RTN-2 is performed.)

## Nested Programs

The ability to nest programs is one of the more useful features of COBOL 85. When programs are nested, an inner program can access resources of an outer program.

The program scope is delineated by the PROGRAM-ID paragraph and the end program header. The end program header, while different in format, is conceptually similar to Scope Terminators.

## External Data

External data is a data item or file that is associated with a run unit rather than with only one program within the run unit. This feature allows the access of data items and files from other compilation units without having to pass them as parameters.

## PERFORM Statement Enhancements

The PERFORM statement executes one or more procedures. When these procedures are finished, control returns to the statement following the PERFORM. The TEST BEFORE and TEST AFTER phrases have been added to the PERFORM statement. The TEST BEFORE phrase causes the condition to be evaluated before the procedures are executed. The TEST AFTER phrase causes the condition to be tested after the procedures are executed. In COBOL 74 , the test was always done before the procedures were executed.

COBOL 85 also supports inline PERFORM statements using the Scope Terminator END-PERFORM. The following example shows an inline PERFORM statement:

```
PERFORM VARYING I FROM 1 BY 1 UNTIL I-4
      READ MY-FILE
      MOVE MY-RECORD TO A(I)
END-PERFORM
```

## CONTINUE Statement

The CONTINUE statement causes an implicit transfer of control to the next executable statement. The following example shows how the transfer is passed:

```
IF condition-1 THEN
    IF condition-2 THEN
        CONTINUE
    ELSE
        PERFORM DO-RTN-1
    END-IF

    PERFORM DO-RTN-2
ELSE
    PERFORM DO-RTN-3 END-IF.

ADD 1 TO XYZ-ITEM.
```

In this example, if condition- 1 and condition-2 are true, DO-RTN-2 is executed. If NEXT SENTENCE were used instead of CONTINUE, control would pass directly to ADD 1 TO XYZ-ITEM.

## EVALUATE Statement

The EVALUATE statement evaluates a set of conditions and, depending on the results of the evaluation, executes one out of several sets of statements. This multibranch statement is similar to CASE statements in other languages.

## CALL Statement

The CALL statement now has the BY CONTENT phrase. When applied to a parameter, this phrase causes a calling program to pass a reference to a copy of the parameter, preventing a called program from modifying the data item whose copy has been passed to it.

## Reference Modification

Reference modification allows access to a portion of a usage display data item without subdividing the item in its data description.

## Intrinsic Functions

The features described in the ANSI X3.23a-1989 addendum to COBOL 85 define the support of many intrinsic functions. This includes mathematical, actuarial, time conversion and other functions. A function is a temporary data item whose value is calculated upon program execution time reference.

## Symbolic Characters

A symbolic character is a user-defined word that specifies a user-defined figurative-constant. This feature of COBOL 85 is similar to a Wang enhancement that allows a hexadecimal character to be defined as a user-figurative-constant.

## Miscellaneous Changes

The following list contains some of the additional changes from COBOL 74 that are either of interest or could have an effect upon existing applications:

<u>Feature</u>	<u>Impact</u>
De-Editing	Data can be moved from an edited data item to an unedited data item.
SET Statement	Condition names can be set to true instead of having to move a value into the associated data item.
RECORD VARYING DEPENDING ON	When this phrase is used in a file entry that contains variable length records, the specified data item contains the number of characters in any given record

Optional Entries For COBOL 85, the following entries are optional:

	ENVIRONMENT DIVISION, CONFIGURATION SECTION, SOURCE-COMPUTER and OBJECTCOMPUTER paragraphs, DATA DIVISION, LABEL RECORD clause, FILLER, and PROCEDURE DIVISION.
Separator	The comma or semicolon followed by a space can now be used interchangeably with the separator space.
INSPECT Statement	The INSPECT statement has a new form that allows each character of a set to be replaced by another character.
Tables	Up to seven dimension tables are permitted.



## ANSI Conformance

The COBOL compiler documented in this manual was developed in accordance with ANSI X3.23-1985 along with X3.23a-1989. (This will subsequently be referred to as the Standard.) The compiler incorporates the features of the Standard, as well as extensions from the *CODASYL COBOL Journal of Development* 1988 and extensions for VS COBOL 74 compatibility that Wang Laboratories, Inc., provides to take full advantage of the operating environment.

A compiler can be in varying degrees of conformance to the Standard, depending upon which of the elements from the various modules have been implemented. There are three levels of conformance: Minimum, Intermediate, and High. Wang COBOL conforms to the high level, which consists of level 2 elements of the Nucleus, Sequential I-O, Relative I-O, Indexed I-O, Inter-Program Communication, Source Text Manipulation, Sort-Merge and Intrinsic Functions modules.

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

No warranty, expressed or implied, is made by any contributor or by any CODASYL COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

The authors and copyright holders of the copyrighted materials used herein, FLOW-MATIC (trademark of Sperry Rand Corporation), programming for the UNIVAC® I and II Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell, have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

## Wang Extensions

COBOL 85 is enhanced with features that support the interactive capabilities and advanced data management facilities available on Wang systems. These enhancements are

- File sharing
- Database support
- Interactive processing
- Program development tools

## **File Sharing Support**

Sharing allows multiple programs to access a resource concurrently and to notify each other of locks held. Hold lock requests are made in units called resources. A resource can be a record, a generic range of keys for an Indexed or alternate Indexed file, or an entire file. Programs can initiate resource requests with either a preclaim strategy, in which all resources are claimed at once, or a claim-as-needed strategy, in which resources are claimed as required. The data management system automatically maintains file consistency and prevents deadlock situations.

Wang COBOL also supports transaction processing capabilities and recovery of indexed data files.

## **Database Support**

The Professional Application Creation Environment (PACE) is a comprehensive application development and information management environment. PACE includes an active data dictionary-driven Relational Data Base Management System (RDBMS) and a set of nonprocedural tools that can increase productivity.

VS programmers can utilize VS PACE services through a Host Language Interface (HLI). COBOL ReSource programmers can utilize PACE for OPEN/ systems services through the Structured Query Language (SQL). These services are supported by pre-processing.

## **Interactive Processing**

In an interactive environment, a user can communicate directly with the system through the workstation. Enhancements to COBOL, supporting this interactive use, allow the writing of programs that can communicate directly with the workstation. COBOL 85 programs can move data to and from the system and workstation screen, format the contents of the screen, and determine the screen's display characteristics.

Workstation and program input-output (I-O) is accomplished by treating the workstation as a file. Two methods support this view of the workstation: Automatic and Code.

Automatic - The DISPLAY AND READ statement moves information to and from the program and the workstation screen, automatically setting default display characteristics, initializing screen fields, and validating data.

Code - Explicit code is required to perform the functions automatically performed by the DISPLAY AND READ statement. The REWRITE statement moves information to the screen, and the READ statement transfers information to the program. Although initially more time consuming, this method allows tighter control of the workstation screen display.

Another extension, the MOVE WITH CONVERSION statement, facilitates the processing of data entered through the workstation. It converts character representation of numbers or numeric data into values that can be used by the program for computation.

## Program Development Tools

Wang COBOL offers a set of easy to use, integrated, system programs to help develop COBOL 85 programs. Three of these programs, the Editor, the Linker and the Symbolic Debugger, are integral to the program development cycle.

### **EDITOR** Utility

The EDITOR utility provides the means for source code entry and modification, and it is integrated with the compiler and the Linker. On the VS, the command driven ADEPT editor also allows advanced editing functions such as concurrent editing of multiple files in separate windows, cut and paste, and pattern matching.

### **LINKER** Utility

The LINKER utility can join any number of programs. Large libraries of useful subroutines and utility programs can significantly cut development time. On the VS, Standard Calling Sequence can link programs written in many different languages. Shared subroutine libraries (SSLs) allow programs to use the most current versions of these subroutines without relinking.

The LINKER utility also allows such functions as section replacement, review of linked section names, and removal of symbolic code from a production program, among others.

### **Symbolic Debugger** Utility

The command driven SYMBOLIC DEBUGGER utility can follow program functions at execution time. The debugger allows the inspection and modification of data values either by symbolic data name or memory address, and it allows monitoring the behavior of a program while it is executing.

# Chapter 2 Program Structure

## Introduction

COBOL programs are organized into a hierarchical structure consisting of programs, divisions, sections, paragraphs, entries, sentences, statements, clauses, and phrases. Divisions are the highest level of the structure; clauses and phrases, the lowest. A COBOL source program can contain other COBOL source programs.

The four divisions are the Identification, Environment, Data, and Procedure divisions. Each division is described in Chapters 3 to 6. The end of a division is indicated by the division header of the next division in the program, by an Identification Division header that indicates the start of another source program, by the END PROGRAM header, or by the end of the source code.

The end of a COBOL program is indicated either by the absence of additional source program lines or by the END PROGRAM header.

### Format

```
Identification-division  
[environment-division]  
[data-division]  
[procedure-division]  
[nested-program]...  
[end program header]
```

## Division Components

Each division in a COBOL program begins with a division header. Except for the Identification Division, a division can have one or more sections, which, in turn, contain paragraphs. (The Identification Division contains only paragraphs.) Depending on the division it is in, a paragraph contains either entries or sentences. An entry contains clauses and phrases, while a sentence contains only phrases.

### Sections

The Environment, Data, and Procedure Divisions contain sections. A section consists of a section header and the related section body. Section headers are terminated by a period. A section ends immediately before the next section or division or at the end of the program.

A section header in the Environment and Data Division is composed of reserved words followed by a period and a space. The Environment Division contains the Configuration section and the Input-Output section. The Data Division contains the File section, the Working-Storage section, and the Linkage section.

A section header in the Procedure Division is composed of a section-name followed by the reserved word SECTION, a period, and a space.

### **Paragraphs**

A paragraph consists of a paragraph header followed by entries or sentences. Each paragraph ends immediately before the next paragraph, section, or division; at the end of the program; or, in the case of the Declaratives section, at the words END DECLARATIVES.

### **Entries**

An entry is a descriptive clause or set of consecutive descriptive clauses terminated by a period. Entries occur in all the divisions except the Procedure Division.

### **Sentences**

A sentence consists of one or more statements terminated by a period. Sentences appear only in the Procedure Division.

### **Statements**

A statement is a syntactically correct combination of words and symbols that begins with a COBOL verb. Statements appear only in the Procedure Division.

### **Clauses**

A clause is a group of words that specifies an attribute of an entry.

### **Phrases**

A phrase is an ordered set of COBOL words, commonly identified with a keyword that forms a portion of a statement or clause.

## **Nested Source Programs**

A COBOL 85 source program can contain other COBOL 85 source programs. Such a contained program is said to be nested. A nested program can, under certain conditions, reference resources of the program that contains it. A nested program has the same organization as any ordinary COBOL program; that is, it is made up of an Identification Division and, optionally, an Environment Division, a Data Division, and a Procedure Division.

A program can be directly or indirectly contained in another program. Suppose program A is contained in program B. If this containment is because program A is contained by some

other program that is, in turn, contained in program B, then program A is indirectly contained in program B. If there is no other program that both contains program A and is contained in program B, then program A is directly contained in program B.

Whenever COBOL source statements follow a nested program, they can be only an END PROGRAM header for the containing program or the Identification Division header for the next contained program. A nested program must be concluded with an END PROGRAM header.

The object code of a compiled nested program is inseparable from the object code of the compiled containing program.

### **PROGRAM-ID of a Nested Program**

The PROGRAM-ID paragraph of a nested program is used to name the program and to assign certain attributes to the program.

#### *Format*

```
COMMON  
PROGRAM-ID. program-name IS    PROGRAM  
INITIAL
```

#### *Syntax Rules*

1. The COMMON clause can be used only if the program is contained in another program.
2. A program-name must conform to the rules for a user-defined word.
3. Program-name-1 cannot be identical to the name of any program that program-1 contains.

#### *General Rules*

1. The COMMON clause specifies that the program can be called by programs other than ones that contain it directly or indirectly.
2. The program-name identifies the source program, the object program, and all listings pertaining to the program.
3. The INITIAL phrase specifies that whenever the program is called, both it and any programs contained within it begin execution in their initial state.

### **END PROGRAM Header**

The END PROGRAM header indicates the end of a source program.

## *General Rules*

1. The END PROGRAM header indicates the end of the source program specified by program-name.
2. If a sequence of source programs in a single source file are to be compiled separately, each program must be terminated by an END PROGRAM header, unless it is the last program in the sequence.

### **Scope of Program-Names**

A program-name can be referenced only by a CALL or CANCEL statement or in an END PROGRAM. Program-names for programs within the same run unit need not be unique. However, when two programs have the same name, at least one of them must be directly or indirectly contained within another separately compiled program that does not contain the other program with the same name.

If a program does not have the common attribute and if it is directly contained in another program, its program-name can be referenced only by statements in the containing program.

For example, if program C has the common attribute and if it is contained in program X, C's program-name can be referenced only by program X and any programs directly or indirectly contained in program X (except for those programs that are also contained in program C).

If a program is separately compiled, its program-name can be referenced by statements in any other program in the run unit, except for programs it directly or indirectly contains.

## Language Structure

The following sections describe the structure of the COBOL language.

### Character Set

The character is the basic unit of the COBOL language. The COBOL character set includes the letters of the English alphabet, Arabic digits, and special characters. The complete set is shown as follows:

#### The COBOL Character Set

Character	Description	Character	Description
0,1,...,9	Digits	,	Comma
A,B,...,Z	Uppercase letters	;	Semicolon
a,b,...,z	Lowercase letters	.	Period
	Space (blank)	"	Quotation mark
+	Plus sign	(	Left parenthesis
-	Minus sign	)	Right parenthesis
	(dash, hyphen)		
*	Asterisk	>	Greater than symbol
/	Slash	<	Less than symbol
	(virgule, stroke)		
=	Equal sign	:	Colon
\$	Currency sign	%	Percent sign



## Character-Strings and Separators

A character-string is a character or sequence of contiguous characters that are a COBOL word, a literal, characters used in the PICTURE clause as symbols defining data categories and editing features, or a comment entry. A separator is a punctuation character (or two contiguous characters considered as an integral unit) that delimits a character-string. The space, comma, semicolon, period, parentheses, quotation mark, and pseudo-text delimiter ( \_ ) are separators.

Separators rules are listed as follows:

1. A period is mandatory wherever it is shown in the formats, and it can be used only where explicitly shown. In the Procedure Division, the period is additionally used to indicate the end of a sentence. A period must always be followed by a space.
2. A space can immediately precede all other separators unless this is expressly prohibited by a rule for a format. However, if a space precedes the closing quotation mark of a nonnumeric literal, it is not a separator but part of the literal.
3. A space can immediately follow all other separators. However, if a space follows the opening quotation mark of a nonnumeric literal, it is not a separator but part of the literal.
4. Anywhere a single space can be used as a separator or as a part of a separator, more than one space can be used.
5. A comma or semicolon immediately followed by a space can be used as a separator anywhere a space can. A space immediately following a comma, semicolon, or period is considered part of that separator and is not a unique separator.
6. Parentheses must be used in pairs when used in subscripts, reference modifiers, arithmetic expressions, or conditions.
7. Pseudo-text delimiters must be used in pairs when used to delimit text. An opening pseudo-text delimiter must be immediately preceded by a space. A closing pseudo-text delimiter must be immediately followed by either a space, comma, semicolon, or period.
8. An opening quotation mark must be immediately preceded by a space or left parenthesis. A closing quotation mark must be immediately followed by either a space, comma, semicolon, period, or right parenthesis.
9. The colon is required when shown in a format and can be used only where shown.
10. A PICTURE character-string must be followed by a space, comma, semicolon, or period.
11. Any punctuation character that appears as part of a PICTURE characterstring or literal is not considered a punctuation character.

*Note: Except where noted, these rules do not apply to nonnumeric literals, comment-entries, or comment lines.*

## Words

A COBOL word is a character-string of not more than 30 characters. The only characters that can be used to make a COBOL word are the letters, the digits, and the hyphen. (A COBOL word cannot begin or end with a hyphen.) A COBOL word is either a reserved word, a system name, or a user-defined word.

A COBOL-defined word is a word that has a defined meaning to COBOL. This meaning is the same in all COBOL programs. COBOL-defined words are either system names, intrinsic function names, or reserved words.

### Reserved Words

A reserved word is one of a special set of words that has a preassigned meaning and can be used only as shown by the formats. There are six types of reserved words: connectives, figurative constants, keywords, optional words, specialcharacter words, and special registers.

**Connectives-** Consist of three types: the logical connectives, AND and OR, which combine conditions; the series connectives, comma and semicolon, which link two or more consecutive operands; and the qualifier connectives, OF and IN, which associate a data name with its qualifier or library name.

**Figurative constants** - Have specific, preassigned values. The singular and plural forms of figurative constants are considered the same, and they can be used interchangeably, the choice usually being made for readability. If figurative constants are put in quotation marks, they become nonnumeric literals. Figurative constants can be used wherever literals can. If the literals must be numeric, the only figurative constants allowed are zeros. The figurative constants are listed as follows:

***ZERO (ZEROS, ZEROES)*** - Represents either the numeric value zero or one or more instances of the alphanumeric character zero, depending on the context.

***SPACE (SPACES)*** - Represents one or more instances of the space character.

***HIGH-VALUE (HIGH-VALUES)*** - Represents one or more instances of the character that has the highest ordinal position in the program collating sequence. If a program collating sequence is the ASCII (American Standard Code for Information Interchange) default, the value of HIGH-VALUE is hexadecimal "FF".

***LOW-VALUE (LOW-VALUES)*** - Represents one or more instances of the character that has the lowest ordinal position in the program collating sequence. If a program collating sequence is the ASCII default, the value of LOW-VALUE is hexadecimal "00".

*Note: Although LOW-VALUE and HIGH-VALUE have hexadecimal values, neither can be used as the sending item for a numeric or numeric-edited receiving item.*

***QUOTE (QUOTES)*** - Represents one or more instances of the quotation character. QUOTE cannot be used to bound a nonnumeric literal or a hexadecimal value. Thus,

QUOTE Wang QUOTE is not a correct way to represent the nonnumeric literal "Wang".

As indicated, a figurative constant represents one or more instances of a character. The number of characters represented is determined from the context as follows:

- When a figurative constant is not associated with another data item, as in a DISPLAY or STOP statement, it represents one character.
- When a figurative constant is moved to or compared with another data item, the number of characters it represents is equal to the size of the other data item (which is not necessarily the number of non-space characters in it).

**Keywords-** Are essential to the meaning of the format in which they appear.

**Optional words-** Improve the readability of source code without affecting the operation of the program.

**Special character words-** Are either arithmetic operators or relational characters. These characters include the plus sign (+), minus sign (-), greater than (>), less than (<), equal sign (=), asterisk (\*), and slash (/).

**Special registers-** Name a compiler-generated storage area that contains information about a COBOL 85 feature. For example, the special register RETURN-CODE is automatically set to a value that indicates the status of an operation.

## System Names

System names are used to communicate with the operating environment. Unless they are also reserved words, system names can be the same as user-defined words.

There are two types of system names: computer names and implementor names. Computer names identify the computer on which the program is compiled and run. The computer name for VS COBOL is WANG-VS and for UNIXI-O COBOL it is WANG-UNIX. Implementor names designate idiosyncratic features of the compiler. For example, SWITCH-1, INDEX AREA, and DATABASE-NAME are Wang implementor names.

## User-Defined Words

User-defined words are ones that the programmer makes up to satisfy the format of a statement or entry. These words are the various program, section, paragraph, and data names. The meaning of user-defined words is unique to the program in which it appears. In the English-like sentences of the Procedure Division, these words serve as nouns.

There are 17 types of user-defined words. Except for level-numbers, each word can belong to only one of these types within a given source program. Most userdefined words must contain at least one alphabetic character. The exceptions are paragraph names, section names, and level numbers. The 17 types of user-defined words are listed as follows:

**Alphabet-name-** Defines a special character set and/or collating sequence. Wang COBOL defaults to the ASCII character set.

**Class-name** - Assigns a name to a group of characters. A data item can be tested to see if it is composed of only characters from the class.

**Condition-name**- Assigns a name to a specific value, set of values, or range of values that a data item can assume. The condition-name is true when the associated data item has one of the defined values and false otherwise.

**Data-name** - Identifies a data item used in the program.

**File-name** - Identifies a file used in the program.

**Index-name** - Associates an index with a table.

**Level-number from 1 through 49** - Denotes the position of a data item within a data structure. Level numbers 66, 77, and 88 identify special properties of an item. A level number less than 10 can be written with or without a leading zero.

**Library-name**- Designates a COBOL library. When a library-name appears in a COPY statement, its text is incorporated in the program when the program is compiled.

**Mnemonic-name**- Identifies a special feature of the system.

**Paragraph-name**- Identifies and marks the start of a paragraph.

**Program-name**- Identifies a source program and the object file entry of that program.

**Record-name** - Identifies a record used in the program.

**Routine-name**- Identifies a procedure written in a language other than COBOL. A routine-name is treated as a comment by Wang COBOL 85 compilers.

**Section-name** -- Identifies and marks the start of a section.

**Symbolic-character** - Specifies a user-defined figurative constant.

**Text-name**- Identifies library text.

**User-figurative-constants**- Are Wang extensions that predate the symbolic character and are designed to serve the same function.

## Literals

A literal is a character-string whose value is determined by the characters of which it is composed. A literal is either nonnumeric or numeric.

### Nonnumeric Literals

A nonnumeric literal is a character-string delimited on both ends by a quotation mark and consists of any allowable characters from the computer's character set. The length of a nonnumeric literal can be from 1 to 160 characters. A single quotation mark in a nonnumeric literal is represented by two contiguous quotation marks. The value of a nonnumeric literal is the string within the quotation marks.

Except for the case of two consecutive quotation marks, any punctuation character in the string is part of the nonnumeric literal rather than a punctuation character.

As a Wang extension, either single or double quotation marks can indicate a nonnumeric literal. If double quotation marks are used, a single quotation mark can be represented in the string without having to be doubled, and vice versa.

### Numeric Literals

A numeric literal is composed of up to 18 digits and, optionally, a plus or minus sign and/or a decimal point. The value of a numeric literal is the algebraic quantity it represents. A numeric literal must contain at least one digit. If it is signed, the sign must be the leftmost character. The rightmost character cannot be the decimal point. An unsigned numeric literal is considered to be positive. If a numeric literal does not contain a decimal point, it is an integer.

*Note: if an otherwise well formed numeric literal is put in quotation marks, it becomes a nonnumeric literal.*

## Reference Format

The COBOL reference format provides a standard method for preparing COBOL source programs. A source program line is defined in terms of the character positions on a workstation screen. Each 80-character line on the terminal corresponds to the first standard input medium, an 80-column punched card. The reference format for a line is shown as follows:

Margin L					Margin C					Margin A					Margin B					Margin R				
1	2	3	4	5	6	7	8	9	10	11	12	13	...											
Sequence Number Area							Area A							Area B										
Indicator Area																								

## Reference Format for a Line

The following list describes the reference format:

**Margin L** - Is immediately to the left of the leftmost character position (position 1) of a line.

**Margin C** - Is between the 6th and 7th character positions of a line.

**Margin A** - Is between the 7th and 8th character positions of a line.

**Margin B** - Is between the 11th and 12th character positions of a line.

**Margin R** - Is to the right of either column 72 or column 80, depending on the setting of the Compiler option R MARGIN. Refer to Appendix B.

There are several areas in a line:

**Indicator area**- Is the 7th character position of a line. The indicator area can contain a symbol that indicates a continuation, debugging, or comment line.

**Area A**- Consists of character positions 8 through 11 (between margin A and margin B). This area is reserved for the beginning of division headers, section names, paragraph names, level indicators, and certain level numbers.

**Area B** - Occupies character positions 12 through margin R. This area contains all the source code that is not permitted in area A.

**Sequence number area**- Consists of the six positions between margin L and margin C. A sequence number identifies a particular line in the source program. Depending on the editor used, this number may be assigned automatically. Sequence numbers are only for convenience and do not affect the compilation of the program.

**Program identification area**- Is columns 73 through 80. An optional string identifying the program can appear in this area. Again, this is a throwback to the days when a dropped box of punched cards would have been a disaster if they could not have been separated by program-ID and then sorted numerically. This area can contain source code if the Compiler option RMARGIN is set to 80. Refer to Appendix B.

## Continuation Lines

In most cases, if a word cannot fit on the rest of a line it can be written on the following line. However, if a character string exceeds one line, it can be continued in area B of the following line, which is then known as a continuation line. A hyphen in the indicator area (column 7) specifies a continuation line.

If the continued character-string is a nonnumeric literal, the first nonblank character in area B of the continuation line must be a quotation mark, followed by the continuation of the literal in the next column. For example:

```
VERY-LONG-AND-COMPLEX-NAME  PICTURE X(26) VALUE IS "ABDCEF  
"GHIJKLMNOPQRSTUVWXYZ"
```

## Comment Lines

A comment line is a line that contains an asterisk or a slash in column 7. Comment lines have no effect on the program logic but are printed in the listing. An asterisk in column 7 causes the line to be printed wherever it appears in the code; the slash causes the comment line to be printed as the first line on the next page.

A comment line can appear anywhere in a program and also in a COBOL library. A comment line can contain any character in the ASCII character set; it is not restricted to characters that can form COBOL words. Successive comment lines are allowed.

As a Wang extension, a comment can also be indicated by the percent sign. Unlike the asterisk and slash, the percent sign can be put in any column of the line. All text from the percent sign to the end of the line is a comment. (When the percent sign is in a nonnumeric literal, it does not indicate a comment.)

## Blank Lines

A blank line is one that is blank from column 7 through margin R. Blank lines can appear anywhere in the source program. They do not affect the compilation and often make the program more readable.

## **Division Headers and Section Names**

The first line in a division must be the division header. It must begin with the division name in area A, followed by a space, the word **DIVISION**, and a period. With the exception of the **USING** phrase of the Procedure Division, no other text can be on the same line as the division header.

A section name must also begin in area A. Like the division header, the section name is followed by a space, the word **SECTION**, and a period. With the exception of the **USE** and **COPY** statements, no other text can appear on the same line as the section header.

A paragraph header and a paragraph name must begin in area A of any line following the first line of a division or section. They are terminated by a period followed by a space. The first sentence or entry of a paragraph can begin either on the same line as the header or name or in area B of a following line. Subsequent sentences or entries can be on the same line or in area B of the following lines. The **END PROGRAM** header must start in area A.

## **Debugging Lines**

A debugging line is a line of code that is intended to assist the programmer in finding errors in the code. Debugging lines predate Interactive Symbolic Debuggers and usually are **DISPLAY** statements that let the programmer know the status of program variables.

## **General Rules**

1. If the **WITH DEBUGGING MODE** clause is specified in the **SOURCECOMPUTER** paragraph, debugging lines are compiled as program lines; if this clause is not present, they are treated as comments. Consequently, the program must be syntactically correct whether the debugging lines are compiled or not compiled.
2. A debugging line is designated by having a 'D' in its indicator area. (A debugging line that consists solely of spaces from margin A to margin R is considered a blank line.)
3. A debugging line is permitted only in a standalone program and must come after the **OBJECT-COMPUTER** paragraph. Successive debugging lines are allowed.

## **Data Description**

To make data as computer independent as possible, COBOL uses a standard format reflecting the data's appearance on a printed page rather than its means of storage. A format uses the COBOL character set to describe data. It also describes the relationship between the way data is defined in the program and the physical characteristics of the storage media.

Data is defined in the Data Division. This data can be referenced and operated on in the Procedure Division. Data is stored in, read from, and written to files. Files are associated with devices such as disks, tapes, workstations, and printers. The following sections describe the format used to define data in COBOL 85 programs.



## Logical and Physical Records

From a programming viewpoint, a file contains uniquely identifiable logical units of data called records. The computer and its input and output devices, however, deal with physical units. Physical records are called blocks. Using blocks makes input-output operations more efficient. Because logical record sizes vary, a block can contain part of one or multiple logical records; and a logical record can be entirely contained in one block or span multiple blocks.

## Buffering

A buffer is a main storage area that the data management system uses when performing input-output operations on behalf of a program. When a file is accessed, blocks of data are taken from the storage device and put into the buffer.

When buffering is effective, a typical READ statement performed by the program retrieves a record from the buffer; and a typical WRITE statement puts a record in the buffer, improving system performance by reducing storage device access.

There are three different buffering schemes:

Default - Allocates a few blocks per buffer.

Large buffer - Can be used with sequential and relative files. It is specified by the RESERVE AREAS clause in the File-Control entry.

Buffer pooling- Shares one buffer area among many files. It is specified by the RESERVE clause in the File-Control entry and the SAME AREA clause in the I-O CONTROL paragraph. Buffer pooling can be used only with indexed files that are opened in I-O mode. Buffering is not available for Shared files due to the needs of record-lock management.

## Data Structure

An elementary data item is described by clauses that indicate characteristics of the data, such as its picture and usage. The concept of a data structure arises from the need to organize data. A data structure is a hierarchical grouping of related data items. Each element of a data structure is either a subordinate data structure or an elementary data item. A data structure is often called a group data item. Referencing a group item accesses all of its subordinate items. A record can be described either as an elementary data item or as a data structure. COBOL uses two types of level numbers to distinguish the items, as explained in the next sections.

### Level Numbers

COBOL uses a system of level numbers to show the relationship between elementary and group items. Since records are the most inclusive data items, a record's level number is always level 01. The items immediately subordinate to the record can have any level number from 02 to 49. Subordinate level numbers do not have to be consecutive, but all data items immediately subordinate to a group item must be described with the same level

number, and this level number must be numerically greater than the level number used to describe the group item.

A group with a given level number includes all group and elementary items that follow it in the description until a level number less than or equal to that group's level number is encountered. For example:

```
01 OPR-A.  
    05 OPR-B PICTURE XXX VALUE "ABC".  
    05 OPR-C.  
        10 OPR-D PICTURE XXX VALUE "DEF".  
        10 OPR-E.  
            15 OPR-F PICTURE XXX VALUE "GHI".  
            15 OPR-G PICTURE XX VALUE "JK".  
    05 OPR-H PICTURE XXXX VALUE "LMNO".
```

The following list shows the contents of each group formatted in the previous example:

Group	Contents
OPR-A	ABCDEFGHIJKLMNO
OPR-B	ABC
OPR-C	DEFGHIJK
OPR-D	DEF
OPR-E	GHIJK
OPR-F	GHI
OPR-G	JK
OPR-H	LMNO

### Special Level Numbers

The special level numbers are described as follows:

66 - Identifies entries describing data items that rename (overlay) other data items.

77 - Identifies noncontiguous data items that are not subdivisions of other items and are not themselves subdivided.

88 - Identifies entries that specify condition names that identify particular values of a conditional variable.

## Classes of Data

There are five categories of data items that are grouped into three classes: alphabetic, numeric, and alphanumeric. The classes and categories are synonymous for alphabetic and numeric items. The alphanumeric class includes alphanumeric edited, numeric edited, and alphanumeric categories. Every elementary item except an index data item belongs to one of the classes and one of the categories. The class of a group item is always alphanumeric, regardless of the class of the elementary items that compose the group item.

The following table shows the relationship between the classes and categories of data items.

**Classes and Categories of Data**

Type of Item	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Alphanumeric Numeric Edited Alphanumeric Edited
Group	Alphanumeric	Numeric Alphabetic Numeric Edited Alphanumeric Alphanumeric Edited

## Representation

The following sections describe how the COBOL compiler represents various values.

### Character Representation

Numeric items whose usage is `DISPLAY` are represented in ASCII characters. Numeric items whose usage is `PACKED-DECIMAL` or `COMPUTATIONAL` are represented in packed decimal format. Items whose usage is `INDEX` are represented as either byte-aligned 16-bit or fullword-aligned 32-bit signed binary numbers, depending on the Compiler option `COMPAT74`. The size of `BINARY` items halfword, fullword, or double word depends on their `PICTURE`. The size of an elementary data item or a group item is the number of bytes in the standard data format of the item. Synchronization and usage can cause a difference between this size and the actual number of characters required for the internal representation.

### Algebraic Signs

There are two types of algebraic signs: *operational* signs, which are associated with signed numeric data items and literals to indicate whether they are positive or negative, and *editing* signs, which appear in reports to identify the sign of the item.

The `SIGN` clause explicitly indicates the location of an operational sign. If it is not used, operational signs for items whose usage is `DISPLAY` are represented as if `SIGN IS TRAILING SEPARATE CHARACTER` or `SIGN IS TRAILING` was specified, depending on the Compiler option `SEPSGN`. Editing signs are indicated by the sign control symbols of the `PICTURE` clause.

### Standard Alignment Rules

When data is moved into an item, the way it is aligned in the item depends on the category of the receiving item. The rules for alignment are

1. If the receiving data item is numeric, the data is aligned by decimal point and is put into the receiving item with zero filling or truncation on either end as required. When there is no explicit decimal point, the data item is considered to have an assumed decimal point immediately following its rightmost character.
2. If the receiving data item is numeric edited, the data is aligned as described for a numeric receiving item. However, editing requirements can cause leading zeros to be replaced by a different character.
3. If the receiving data item is alphanumeric, alphanumeric edited, or alphabetic, the *leftmost* character of the incoming data is aligned with the *leftmost* character position in the receiving data item. Space fill or truncation on the right occurs if the data is not the same size as the receiving item. However, if the `JUSTIFIED` clause is specified for the receiving item, alignment is by the rightmost characters.

## Scope of Names and Uniqueness of Reference

The scope of user-defined names is restricted to the program in which they are declared. When a program is contained (directly or indirectly) within another program, it may refer to user-defined names declared in the containing program with the GLOBAL attribute. If a contained program describes data items with names identical to GLOBAL names, any references to these names within the contained program are references to local entities.

To reference an item, its name must be unique. A nonunique user-defined name explicitly referenced in the program must be made unique through qualifiers that preclude any ambiguity of reference. Qualification and subscripting are ways of ensuring uniqueness of reference. A data-name that is made unique by any necessary qualification and subscripting is called an identifier.

### Qualification

Qualification ensures uniqueness of reference of an item by associating the item with an item that contains it. Data-names, even subscripted or indexed ones; conditional variables; and paragraph names can all be made unique by qualification. To reference an item by using qualification, a data-name, condition-name, paragraph-name, or text-name is followed by one or more phrases composed of a qualifier preceded by IN or OF.

A data-name can be qualified if it is part of a hierarchy. In a hierarchy, names associated with a level indicator are the most significant; they must be unique. The next most significant are names associated with level-number 01, followed by level-number 02, and so on. A reference to the data-name can be made unique by qualifying it with one or more of the names of items at the higher levels of the hierarchy.

However, a data-name cannot be both the name of an external data item and the name of another external data item within a contained or containing program. Nor can a given data-name be both the name of a global item and the name of any other data item described in the program that describes that global item.

The following additional restrictions apply to the declarations of data items with the EXTERNAL or GLOBAL attributes:

- One data-name cannot be used in both a containing program and a contained program to describe data items that have the EXTERNAL attribute in both programs.
- A paragraph-name can be qualified by the name of the section that contains it.
- Regardless of the available qualification, no name can be both a data-name and a procedure-name.
- The name of a conditional variable can be used as a qualifier for any of its condition-names.

## Formats for Qualification

The following examples show formats used for qualification.

### **Format 1**

$$\text{data-name-1} \left[ \left\{ \left\{ \begin{array}{c} \text{IN} \\ \text{OF} \end{array} \right\} \text{data-name-2} \right\} \dots \left[ \left\{ \begin{array}{c} \text{IN} \\ \text{OF} \end{array} \right\} \text{file-name-1} \right] \right]$$

### **Format 2**

$$\text{condition-name-1} \left[ \left\{ \begin{array}{c} \text{IN} \\ \text{OF} \end{array} \right\} \text{file-name-1} \right]$$

### **Format 3**

$$\text{paragraph-name-1} \left[ \left\{ \begin{array}{c} \text{IN} \\ \text{OF} \end{array} \right\} \text{section-name-1} \right]$$

### **Format 4**

$$\text{text-name-1} \left[ \left\{ \begin{array}{c} \text{OF} \\ \text{IN} \end{array} \right\} \text{library-name-1} \left[ \left\{ \begin{array}{c} \text{OF} \\ \text{IN} \\ \text{ON} \end{array} \right\} \text{volume-name-1} \right] \right]$$

### **Format 5**

LENGTH OF identifier

### **Format 6**

ADDRESS OF identifier

## General Rules

1. A name can be qualified, even if it is already unique. When more than one combination of qualifiers ensures uniqueness, any one of them can be used.
2. Qualifiers are specified in the order of successively more inclusive levels in the hierarchy. When specifying qualifiers, IN and OF are equivalent.
3. Each qualifier used in format 1 must be a name associated with a level indicator, the name of a group item to which the item being qualified is subordinate, or the name of the conditional variable with which the condition-name being qualified is associated.
4. Data-name-1 or data-name-2 can be a record-name.
5. A paragraph-name need not be qualified when referenced from within the section that contains it. When a paragraph is explicitly referenced, its name must be unique within the section, because a section-name is the only way to qualify a paragraph-name. When a paragraph-name is referenced from another section, the reserved word SECTION cannot appear in the qualifier.
6. Unless the Compiler option COPYLIBS is YES (refer to Appendix B), a text-name must be qualified each time it is referenced if more than one COBOL library is available.

7. The special register LINAGE-COUNTER must be qualified each time it is referenced if more than one file description entry in the source program contains a LINAGE clause.
8. The words LENGTH OF can be used to qualify a data-name. When this Wang extension is used, the length of the data-name in bytes is used in the operation instead of the value of the data-name.

The combination LENGTH OF *data-name* is considered to have a usage of binary and a PICTURE of 9(9). It can be used anywhere that an integer can be used- in particular, as the sending item of a MOVE statement, in the FROM and BY phrases of a PERFORM statement, and in the USING phrase of a CALL statement, where it is always passed by content.

If *data-name* is a group item, any implicit FILLER is included in its length. If it is reference modified, the length of the specified substring is used in the operation.

9. The words ADDRESS OF can be used to qualify a data item. When this Wang extension is so used, the *address* of the item is used instead of the contents of the data-name being used in the operation.

The ADDRESS OF data-name is treated the same as a data item with a usage of POINTER, and it can be used in a conditional expression or in the SET statement. However, if the ADDRESS OF data-name is the receiving item of a SET statement, the data-name must be the name of a based record, that is, a record declared in the Linkage section of the Data Division that is not referenced in the USING clause of the Procedure Division and cannot be subscripted or reference modified.

## Subscripting and Indexing

Because table elements do not have individual data names, a subscript is used to reference an individual element of a table. The subscript can be either a dataname or a numeric literal that is an integer. If it is a data-name, the data item must be a numeric integer. In the program, subscript(s) are written in parentheses following the data-name.

An index is a special type of subscript. The value of an index is an occurrence number that denotes a table element. The advantage of indexing over normal subscripting is that it facilitates the searching of tables. To use indexing, one or more indexes must be associated with a table in the INDEXED BY clause.

For more information on table handling, subscripting, and indexing, refer to the sections titled "SEARCH Statement" and "SET Statement" in Chapter 7.

### Format

$$\left\{ \begin{array}{l} \text{condition-name-1} \\ \text{data-name-1} \end{array} \right\} ( \left\{ \begin{array}{l} \text{integer-1} \\ \text{data-name-2} \ [ \ \{\pm\} \ \text{integer-2} \ ] \\ \text{index-name-1} \ [ \ \{\pm\} \ \text{integer-3} \ ] \end{array} \right\} \dots )$$

### *Syntax Rules*

1. The data description entry containing data-name-1 or the data-name associated with condition-name-1 must contain an OCCURS clause or must be subordinate to a data description entry that contains an OCCURS clause.
2. The number of subscripts used to reference a table element must equal the number of OCCURS clauses in the description of that element. When more than one subscript is required, the subscripts are written in the order of successively less inclusive dimensions of the table. However, subscripting is not allowed when the element is the subject of a SEARCH statement, is in a REDEFINES clause, or is in the KEY IS phrase of an OCCURS clause.
3. If a table description contains an INDEXED BY index-name-1 phrase, index-name-1 must be a data description entry in the table's hierarchy.
4. If integer-1 is signed, it must have a positive value.
5. Data-name-2 can be qualified; it must be a numeric integer.

### *General Rules*

1. The value of the subscript must be a positive integer. The first element of a table is an element. The highest permissible subscript value is the maximum number of occurrences specified by the OCCURS clause.
2. The value of index-1 must correspond to the occurrence number of an element in its associated table. Index-1 must be initialized before it is used as a subscript. This initialization can be done by a PERFORM statement with the VARYING phrase, a SEARCH statement with the ALL phrase, or a SET statement.
3. If integer-2 or integer-3 is specified, the value of the subscript is determined by using these values to increment or decrement the value of index-1 or of data item-2.
4. If the Compiler option SUBCHK is YES (refer to Appendix B), the object program contains code that checks the ranges of subscripts whenever a table element is accessed and causes a program to interrupt if a subscript exceeds its valid boundaries.

## **Condition-Names**

A condition-name specifies a unique value, a set of values, or a range of values that a data item can have. A data item that has an associated condition-name is called a conditional variable. Each condition-name must be unique or be able to be made unique through qualification and/or subscripting.

If qualification is used to make a condition-name unique, the associated conditional variable can be used as the first qualifier. The hierarchy of names associated with the conditional variable can also be used to make the condition-name unique.



If references to a conditional variable require subscripting, references to any of its condition-names also require subscripting. When condition-name appears in this manual, it refers to a condition-name that can be qualified or subscripted, if necessary to ensure uniqueness.

## Corresponding Data Items

A pair of data items are corresponding if

- Each is subordinate to a different group item.
- They have the same data-name and qualifiers up to, but not including, the group item names.
- Neither of the items is a FILLER.
- Neither data item has a REDEFINES, OCCURS, or USAGE IS INDEXED clause in its description nor is subordinate to a data item with one of these clauses.
- The description of neither group item contains level-number 66.
- The CORRESPONDING phrase can be used in the ADD, SUBTRACT, or MOVE statement. It allows the addition, subtraction, or moving of data items from one group item to another without having a separate statement for each data item.
- In a MOVE statement, at least one of the corresponding data items must be an elementary item.
- In an ADD or SUBTRACT statement, both of the corresponding data items must be numeric elementary items.

## Reference Modification

Reference modification references only a portion of a data item. To access an item using reference modification, the position and length of the substring within the data item are specified.

Format

data-name-1 ( leftmost-character-position : [ length ] )

*Syntax Rules*

1. The leftmost-character-position and length must be arithmetic expressions
2. Unless otherwise specified in the rules for a statement, reference modification can be used anywhere an alphanumeric data item can be specified.

3. Data-name-1 can be qualified and/or subscripted.
4. Data-name-1 must have a usage (explicit or implicit) of DISPLAY.

### *General Rules*

1. Data-name-1 must have the USAGE DISPLAY attribute, explicit or implicit.
2. Each character of data-name-1 is assigned an ordinal number, starting with the leftmost character, which is assigned the number one. (If the description of data-name-1 contains a SIGN IS SEPARATE clause, the sign is also assigned a number.)
3. The data item that results from reference modification is treated like an alphanumeric elementary data item without the JUSTIFIED clause.
4. Reference modification for an operand is evaluated after the evaluation of any subscripts.
5. Reference modification creates a unique data item determined as follows:

The expression for the leftmost character position is evaluated to determine the first character of the item. This evaluation must result in a positive integer that is less than or equal to the number of characters in data-name-1.

The expression for the length is evaluated to determine the number of characters in the item. This evaluation must result in a positive integer. In addition, the sum of the leftmost character position and length, minus one, must be less than or equal to the number of characters in data-name-1. If a length is not specified, the item consists of all the characters from the one specified to the rightmost character of data-name-1.

### *Example*

```
01  PERSONNEL-RECORD .  
    05  NAME          PICTURE X(30) .  
    05  ADDRESS       PICTURE X(60) .  
  
    MOVE ADDRESS (56:5) TO ZIP-CODE-DISPLAY .
```

The MOVE statement moves the last five characters of ADDRESS into ZIP-CODE-DISPLAY. (This example assumes that the ZIP code is in the last five positions of ADDRESS.)

# Chapter 3 Identification Division

## Introduction

The Identification Division is always the first division in a program, and it is the only required division. In addition to containing the paragraph that names the program, it can also have optional paragraphs that aid in documenting the program. These entries are considered comments.

### **Format**

IDENTIFICATION DIVISION.

PROGRAM-ID. program-name  $\left[ \text{IS } \left\{ \begin{array}{c} \text{COMMON} \\ \text{INITIAL} \end{array} \right\} \text{PROGRAM} \right].$

[ AUTHOR. [ comment-entry ] ... ]

[ INSTALLATION. [ comment-entry ] ... ]

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

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

[ SECURITY. [ comment-entry ] ... ]

## PROGRAM-ID Paragraph

The PROGRAM-ID paragraph contains the name of the program.

### **Format**

PROGRAM-ID. program-name  $\left[ \text{IS } \left\{ \begin{array}{c} \text{COMMON} \\ \text{INITIAL} \end{array} \right\} \text{PROGRAM} \right].$

### *Syntax Rule*

A program-name must follow the rules for a user-defined word.

### *General Rules*

1. The program-name is the name by which the object module and associated listings are identified. In addition, it is used if the program is called by another program.
2. When the INITIAL phrase is included, whenever the program is called, both it and any programs contained within it begin execution in their initial state.
3. When the COMMON clause is included, the program can be called by programs other than ones that contain it directly or indirectly.

## Initial State of a Program

The initial state of a program is the state that a program, together with its associated data items, is in when it begins execution the first time it is called. When a program is in its initial state

- All internal (Working-Storage) data items with a VALUE clause are set to that value.
- Internal files are closed.
- The control mechanisms for all PERFORM statements are set to their initial states.
- All GO TO statements referred to by ALTER statements in the same program are reset to their initial destinations.

If a program has the INITIAL attribute, it is in its initial state every time it is called. Otherwise, it is in its initial state only when it is the first time it is called

- During execution of the run unit
- After the execution of a CALL statement referencing any other program that has the INITIAL attribute and directly or indirectly contains it
- After the execution of a CANCEL statement for it or for some other program that directly or indirectly contains it

## Common Programs

A common program is one that can be called by any program directly or indirectly contained in the program that contains it, except those programs contained in itself. In other words, if program X, which is contained in program Y, is a common program, it can be called by any program directly or indirectly contained in program Y as long as the program is not contained in program X.

A program has the common attribute when it has the COMMON clause in its PROGRAM-ID paragraph.

## Additional Paragraphs

Any of five additional paragraphs can be used to document the Identification Division: AUTHOR, INSTALLATION, DATE-WRITTEN, DATECOMPILED, and SECURITY. Their format is shown in one group at the beginning of this chapter.

### *Syntax Rules*

1. A comment-entry can contain any character in the computer's character set.
2. The comment-entry can be contained no more than one line. When a comment-entry takes more than one line, the hyphen cannot be used in the indicator area.
3. The optional paragraphs can appear in any order.

### *General Rule*

In the compilation source listing, the DATE-COMPILED paragraph is replaced by

DATE-COMPILED. current date.

# Chapter 4 Environment Division

## Introduction

The Environment Division describes the computer environment in which the program will run. This division is optional; but if it is included, it must come directly after the Identification Division. The Environment Division has two sections: the Configuration section and the Input-Output section

The following formats of the sections and paragraphs in the Environment Division define the order in which they must appear in the source program.

Format

```
[ENVIRONMENT DIVISION.  
[CONFIGURATION SECTION.  
[ SOURCE-COMPUTER. [source-computer-entry] ]  
[ OBJECT-COMPUTER. [object-computer-entry] ]  
[ SPECIAL-NAMES. [special-names-entry] ]  
[ FIGURATIVE-CONSTANTS. [user-figurative-constant-entry] ] ]  
[INPUT-OUTPUT SECTION.  
[ FILE-CONTROL. { file-control-entry } ...  
[ I-O-CONTROL. [ input-output-control-entry ] ] ] ] ]
```

## Configuration Section

The Configuration section identifies the computer(s) on which a program is compiled and executed. It contains the SPECIAL-NAMES and FIGURATIVECONSTANTS paragraphs.

A Configuration section cannot appear in a program that is contained within another program.

The entries explicitly or implicitly stated in the Configuration section of a program apply to each program contained within that program.

## SOURCE-COMPUTER Paragraph

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

### Format

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

### *General Rules*

1. The computer-name is treated as a comment. If specified, on VS systems it should be WANG-VS and on UNIX systems, WANG-UNIX. When a computer-name is not specified, the compiler assigns the name of the computer upon which the program is being compiled.
2. The WITH DEBUGGING MODE clause indicates how the compiler will treat debugging lines. When present, debugging lines are compiled; when absent, debugging lines are considered comment lines.

## OBJECT-COMPUTER Paragraph

The OBJECT-COMPUTER paragraph identifies the computer that executes the program and, optionally, identifies the collating sequence the program uses.

### **Format**

```
OBJECT-COMPUTER. [computer-name [ MEMORY SIZE integer-1 { CHARACTERS  
WORDS  
MODULES } ] ]  
[ PROGRAM COLLATING SEQUENCE IS alphabet-name-1 ] ].
```

### *General Rules*

1. All clauses in this paragraph also apply to any contained programs.
2. The computer-name is treated as a comment. If specified, on VS systems it should be WANG-VS and on UNIX systems, WANG-UNIX. The compiler assigns the name of the computer upon which the program is being compiled.
3. The MEMORY SIZE clause is treated as a comment.
4. If the PROGRAM COLLATING SEQUENCE clause is specified, the program collating sequence is the one indicated by alphabet-name-1. Otherwise, the program collating sequence is the ASCII collating sequence.

The program collating sequence is used to determine the truth value of nonnumeric comparisons in relation conditions or in condition-name conditions. It is also used in comparing nonnumeric keys in MERGE or SORT statements, unless the statement contains a COLLATING SEQUENCE phrase.

## SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph relates names specific to the operating system to mnemonic-names, alphabet-names to character sets or collating sequences, and class-names to sets of characters. In addition, it can change the characters used to represent the currency sign and the decimal point.

### Format

```
SPECIAL-NAMES [ [ { WORKSTATION
                  ONE-LINE
                  SWITCH-X
                }
                [ IS mnemonic-name-1 [ON STATUS IS condition-name-1 [OFF STATUS IS condition-name-2]]
                [ IS mnemonic-name-1 [OFF STATUS IS condition-name-2 [ON STATUS IS condition-name-1]]
                [ ON STATUS IS condition-name-1 [ OFF STATUS IS condition-name-2]
                [ OFF STATUS IS condition-name-2 [ ON STATUS IS condition-name-1]
                ] ] ...
                [ ALPHABET Alphabet-name-1 IS { STANDARD-1
                                                STANDARD-2
                                                NATIVE
                                                { literal-1 [ THROUGH literal-2 ] } ...
                                                [ ALSO literal-3 ] . } ...
                ] ...
                [ SYMBOLIC CHARACTERS { {Symbolic-character-1} { IS
                                                ARE
                                                } {integer-1} ... } ...
                [ IN alphabet-name-2 ] ...
                [ CLASS class-name-1 IS {literal-4 [ THROUGH literal-5 ] } ... ] ...
                [ CURRENCY SIGN IS literal-6 ]
                [ DECIMAL-POINT IS COMMA ]
```

### Syntax Rules

1. Up to seven switches, named SWITCH-1 through SWITCH-7, can be used.
2. A mnemonic-name associated with WORKSTATION can be used only in an ACCEPT statement.
3. A mnemonic-name associated with ONE-LINE can be used only in the ADVANCING clause of a WRITE statement.
4. The ON STATUS and OFF STATUS clauses can be written in any order.
5. If a literal in the ALPHABET clause is numeric, it must be an unsigned integer with a value from 1 through 256.
6. If a literal in the ALPHABET clause is nonnumeric and associated with a THROUGH or ALSO phrase, it must be only one character long.
7. A given character cannot be specified more than once in the ALPHABET clause.
8. Literal-1 through literal-5 must not specify a symbolic-character figurative constant.
9. In the SYMBOLIC CHARACTERS clause, the number of symboliccharacters must equal the number of integers. The relationship between each symbolic-character-1 and



the corresponding integer is by position in the clause, the first symbolic-character being paired with the first integer, and so forth.

If the IN phrase is not specified, the position specified by integer-1 must exist in the native character set. If the IN phrase is specified, the position must exist in the character set named by alphabet-name-2.

10. A given symbolic-character can appear only once in the SYMBOLIC CHARACTERS clause.
11. Literal-6 must not be a figurative constant.

### *General Rules*

1. All clauses specified in the SPECIAL-NAMES paragraph for a program also apply to programs contained within that program. A condition-name specified in the containing program's SPECIAL-NAMES paragraph can be referenced from any contained program.
2. The switches are external switches. Whenever a program using switches is run, a screen listing the switch numbers appears on the workstation, allowing the user to turn the switches on or off.
3. If the on or off status of a SWITCH is associated with a condition-name, the status of the switch can be determined by testing those conditionnames.
4. A mnemonic-name associated with a SWITCH can be used *only* in a SET statement.
5. The ALPHABET clause provides a means of relating a name to a specified character code set and/or collating sequence. In the ALPHABET clause, STANDARD-1, STANDARD-2, and NATIVE all specify the American Standard Code for Information Interchange (ASCII) X3.4-1977 character set or collating sequence.

When alphabet-name-1 is referenced in either the PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph, or in the COLLATING SEQUENCE phrase of a MERGE or SORT statement, it specifies a collating sequence. When referenced in a CODE-SET clause in a file description entry, it is considered a comment. However, to conform with the Standard, if the literal phrase is specified, the alphabet-name cannot be used in a CODE-SET clause.

6. The collating sequence specified in the ALPHABET clause has the following properties:
  - The order in which the literals appear in the ALPHABET clause specifies, in ascending sequence, the ordinal number of the characters within the collating sequence.
  - If a literal is numeric, it specifies the ordinal position of a character in the native character set. This value cannot exceed 256.

*Note: The ordinal positions start with position 1. This position contains the character whose ASCII code value is "00".*

- If a literal is nonnumeric, it specifies the actual character in the native character set. If this nonnumeric literal contains more than one character, each character, beginning with the leftmost, is assigned a successive ascending position in the collating sequence.
  - Any characters in the native collating sequence that are not explicitly specified in the literal phrase assume a position greater than any of the explicitly specified characters. The relative order of these unspecified characters is the same as in the native collating sequence.
  - If the THROUGH phrase is specified, each contiguous character in the native character set, beginning with literal-1 and ending with literal-2, is assigned a successive ascending position in the collating sequence. These characters can be specified in an ascending or descending sequence.
  - If the ALSO phrase is specified, the characters of the native character set specified by literal-1 and literal-3 are assigned to the same ordinal position in the collating sequence being specified. If alphabet-name-1 is referenced in a SYMBOLIC CHARACTER clause, only literal-1 is used to represent the character in the native character set
7. When specified as literals in the SPECIAL-NAMES paragraph, the figurative constants HIGH-VALUE and LOW-VALUE are associated with those characters having the highest and lowest positions in the native collating sequence. When these figurative constants are used elsewhere, the character that has the highest ordinal position in the program collating sequence is associated with HIGH-VALUE, and the character that has the lowest ordinal position is associated with LOW-VALUE. If more than one character has t&ehw\$hest, or km eApossaon, the fXn~ chacactec spec~f~ed ~s associated with the respective figurative constant.
  8. If the IN phrase is not specified in the SYMBOLIC CHARACTERS clause, symbolic-character-1 represents the character whose ordinal position in the native character set is specified by integer-1. If the IN phrase is specified, integer-1 specifies the ordinal position of the character that is represented in the character set named by alphabet-name-2.
  9. The internal representation of symbolic-character-1 is the internal representation of the character in the native character set.
  10. The CLASS clause defines class-name-1 as consisting of the specified characters. Class-name-1 can be used only in a class condition. The specifications are listed as follows:
    - If a literal is numeric, it specifies the ordinal position of a character in the native character set. This value cannot exceed 256.

- If a literal is nonnumeric, it specifies the actual character in the native character set. If the nonnumeric literal contains more than one character, each character in the literal is included in the class.
  - If the THROUGH phrase is specified, the contiguous characters in the native character set are included in class-I, beginning with literal-1, and ending with literal-2. These characters can be in either ascending or descending sequence.
11. Literal-6 in the CURRENCY SIGN clause is used in a PICTURE clause to represent the currency symbol. If this clause is not specified, the dollar sign (\$) is the currency sign.
  12. This literal must be nonnumeric and a single character. It can be any character in the native character set, except for the
    - Digits 0 through 9
    - Uppercase letters A, B, C, D, P, R, S, V, X, Z
    - Lowercase letters a through z and the space character
    - Special characters \* + - , . ; ( ) " ' = /
  13. When the DECIMAL-POINT IS COMMA clause is specified, the functions of the comma and decimal point are reversed in the characterstring of the PICTURE clause, in numeric literals, in data operated on by a MOVE WITH CONVERSION statement, and in fields of a DISPLAY-WS record.

## FIGURATIVE-CONSTANTS Paragraph

A figurative constant is a reserved word that has a specific value. Several figurative constants are built into the COBOL language, such as HIGH-VALUE, ZEROES, and QUOTE. The FIGURATIVE-CONSTANTS paragraph is a Wang extension that associates figurative constants specific to the program with hexadecimal values. These figurative constants can be used to control the order area and field attribute characters (FACs) of the display or to control printer functions such as sounding the alarm and advancing lines.

*Note: Appendix C has information about FAC's, Appendix D has information about the workstation screen order area, and Appendix F has information about printer control functions.*

### Format

FIGURATIVE-CONSTANTS . [ data-name-1 IS "hex-value" ] ...

### Syntax Rules

1. Hex-value is composed of hexadecimal digits 0 to 9 and A to F.

2. Either two or four hexadecimal digits (one or two bytes) can be defined as hex-value. This value must be enclosed in quotes.
3. The value of data-name-1 is the specified hexadecimal value.

### *General Rules*

1. If a figurative constant data item that has a 2-digit hexadecimal value is used in a VALUE clause, or as the object of a MOVE statement, the receiving item is completely filled with the specified value.
2. If a figurative constant data item that has a 4-digit hexadecimal-value is used, it acts as a 2-character literal. If the receiving item of a MOVE statement or a VALUE clause is longer than two characters, the remaining positions are space-filled on the right.

## **Input-Output Section**

The Input-Output section is an optional part of the Environment Division that contains information necessary for the transfer of data between external media, such as disks and workstations, and the object program. The Input-Output section contains two paragraphs: FILE-CONTROL and I-O-CONTROL paragraphs.

### Format

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

## **FILE-CONTROL Paragraph**

The FILE-CONTROL paragraph contains information about the files used in the program. Each file is described by a file-control entry. The file-control entry gives a file a name and specifies certain attributes of the file.

### **Format 1 – Sequential File**

```
SELECT [ OPTIONAL ] file-name

ASSIGN TO { "parameter-reference-name" }
          { data-name-1 }

          [ "DISK"
            "DISPLAY"
            "PRINTER"
            "TAPE"
            "STANDARD-DISK" ]

          [ NODISPLAY ] [ [NO] RESPECIFY ]
          [ DISPLAY ] [ NORESPECIFY ]

[ RESERVE integer-1 ] [ AREA
                       AREAS ]

[ BUFFER SIZE IS integer-2 BLOCKS ].

[ FILE STATUS IS data-name-2 ]

[ [ ORGANIZATION IS ] SEQUENTIAL ]

[ ACCESS MODE IS { SEQUENTIAL
                   { RANDOM
                   { DYNAMIC
                   } } } RELATIVE KEY IS data-name-3 ] ]

[ CURSOR POSITION IS data-name-4 ]

[ PFKEY IS data-name-5 ]

[ PADDING CHARACTER IS { data-name-6
                        literal-1
                        } ]

[ RECORD DELIMITER IS { STANDARD-1
                        data-name-7
                        } ]
```

## **Format 2 – Relative File**

```
SELECT [ OPTIONAL ] file-name-1  
ASSIGN TO { "parameter-reference-name" }  
           { data-name-1  
             [ "STANDARD-DISK"  
               "DISK"  
               [ NODISPLAY ] [ [NO] RESPECIFY ]  
               [ DISPLAY ] [ NORESPECIFY ]  
             [ RESERVE integer-1] [ AREA  
                                     AREAS ] ]  
[ BUFFER SIZE IS integer-2 BLOCKS ].  
[ FILE STATUS IS data-name-2 ]  
[ ORGANIZATION IS ] RELATIVE  
ACCESS MODE IS { SEQUENTIAL  
                  RANDOM  
                  DYNAMIC } RELATIVE KEY IS data-name-3 ] ]
```

### Format 3 – Indexed File

```
SELECT [ OPTIONAL ] file-name-1
ASSIGN TO { "parameter-reference-name"
            data-name-1
            [ "STANDARD-DISK"
              "DISK"
              [ NODISPLAY ] [ [NO] RESPECIFY ]
              [ DISPLAY ] [ NORESPECIFY ]
            ]
[ RESERVE integer-1] [ AREA
                       AREAS ] ]
[ BUFFER SIZE IS integer-2 BLOCKS ].
[ FILE STATUS IS data-name-2 ]
[ ORGANIZATION IS ] INDEXED [ DMS
                                XDMS ]
                                [ WITH HASHSIZE OF integer-3 BLOCKS ]
                                [ ORDER BY { KEY
                                              ENTRY } ]
[ ACCESS MODE IS { SEQUENTIAL
                   RANDOM
                   DYNAMIC
                 } RECORD KEY IS data-name-3 ] ]
[ ALTERNATE RECORD KEY([integer-4]IS data-name-4[WITH DUPLICATES])... ]
```

### **Format 4 – Sort-Merge File**

```
SELECT file-name-1  
  
      ASSIGN TO { "parameter-reference-name" } [ "TAPE" ]  
               { data-name-1 } [ "DISK" ]  
  
      [ NODISPLAY ] [ [ NO ] RESPECIFY ]  
                   [ NORESPECIFY ]  
  
      [ RESERVE integer-1 ] [ AREA ]  
                           [ AREAS ] ]  
  
[ BUFFER SIZE IS integer-2 BLOCKS ].
```

#### Syntax Rules

1. The SELECT clause (see Format-1 - Sequential File) must come first; the remaining clauses can be in any order.
2. Each file described in the Data Division must also be specified in the F10LECONTROL paragraph. Conversely, a file specified in this entry must have a file description entry in the Data Division.
3. If OPTIONAL is specified for a file, the file can only be opened in INPUT, Special Input, Shared, I-O, or EXTEND mode.
4. File-name-1 is the name used in FD or SD entries and in Procedure Division statements that reference the file.
5. Data-name-1 must be alphanumeric; it can be qualified.
6. The device type must be in quotes. If nothing is specified, DISK is assumed.
7. Integer-1 indicates the number of blocks to be allocated; it must be from 1 to 9 for sequential and relative files, and from 3 to 255 for indexed files.

#### General Rules

1. The first eight characters of parameter-reference-name (which must be written in quotes) or data-name-1 (see Format 1 - Sequential File) is a label used to obtain a filename from the user at runtime. When the file is opened, the specified value is displayed at the user's workstation, and the user then supplies the filename in a blank field which is displayed next to the label. The first character of the filename must be alphabetic; the name cannot exceed eight characters. If the file is a workstation file, the entry typed in must be "SCREEN" (including the quotes).
2. The NODISPLAY clause suppresses the OPEN prompt at the user's screen at runtime. This prompt is unnecessary if the physical file identification information is available either through a procedure or in the VALUE OF clause in the file description entry.



3. When RESPECIFY is specified, a respecification prompt is issued by the data management file system if an error occurs that precludes opening the file. This respecification prompt gives the user another chance to name the physical file.

When NO RESPECIFY (or NORESPECIFY) is used, and the file cannot be opened, the associated file status variable is updated to indicate the nature of the error. In addition, if there is an applicable declarative procedure, it is executed. If no declarative procedure exists, subsequent action depends on the setting of the COMPAT74 switch.

If the Compiler option CONVAT74 is YES or I-O, control is returned to the statement following the OPEN statement. If COMPAT74 is NO or NUC, a runtime error occurs and the program must be canceled.

4. When the device type assignment is "TAPE", the ACCESS MODE must be sequential.
5. Integer-1 indicates the number of I-O blocks to be allocated. For indexed files, it must be from 3 to 255; for all other files, including sort-merge files, it must be from 1 to 9.
6. The BUFFER SIZE clause functions the same as the RESERVE clause. It is included for compatibility with Wang VS COBOL 74.
7. The BUFFER SIZE clause specifies the size of the buffer for input-output operations in blocks. If neither this clause nor the RESERVE clause is specified, the default is one block for non-indexed files and two blocks for indexed files.

## FILE-CONTROL Clauses

The following FILE-CONTROL entry clauses are presented in the order of their occurrences in the preceding general formats.

### ORGANIZATION Clause

The ORGANIZATION clause specifies the logical structure of a file.

#### *General Rules*

1. A file's organization cannot be changed after the file has been created, and matching organization is verified when opening an existing file.
2. The DMS or XDMS clause affects file creation only on VS systems that have XDMS (Extended DMS) installed. This clause is closely related to the VS COBOL 85 Compiler option FILES, discussed in Appendix B.
3. The ORDER BY KEY clause specifies that records having duplicate alternate keys are ordered according to the value of their primary keys. ORDER BY ENTRY specifies that records having duplicate alternate keys are ordered in the sequence in which they were entered in the file. ORDER BY ENTRY cannot be selected for VS DMS files but can be selected for any Resource Indexed file.

4. The HASHSIZE clause applies to VS XDMS only; it cannot be selected for VS DMS files. It is treated as a comment on UNIX COBOL. This clause specifies that the internal organization of an Indexed file uses hash tables instead of binary trees. Integer-3 specifies the number of hash table blocks to be collected.

*Note: The only form of the START statement that can be used with hashed files is START EQUAL. Also, if a READ NEXT is done on a hashed file, the record obtained is undefined. In COBOL Resource, the HASHSIZE clause is ignored because Resource (PDMS) does not support hashed indexes. An index specified as a hashed index is created as a regular (nonhashed)index.*

## ACCESS MODE Clause

The ACCESS MODE clause specifies the way in which records are accessed.

### *Syntax Rules*

1. If this clause is omitted, sequential access is assumed.
2. A file that is used in the USING or GIVING phrase of a SORT or MERGE statement cannot have an ACCESS MODE of RANDOM.
3. IF RANDOM and DYNAMIC access is specified, a key must be specified for the file.
4. If the device type is "TAPE", the ACCESS MODE must be sequential.
5. If a Relative file is to be referenced in a START statement, it must have a RELATIVE KEY phrase.
6. For a Consecutive or a Relative file, data-name-3 must be an unsigned integer whose picture does not contain the symbol P. It cannot be defined in a record description entry associated with the specified file.
7. For an Indexed file, data-name-3 must be defined within a record description entry associated with the specified file and can be qualified.

If the file contains more than one record description entry, it is necessary to describe data-name-3 in only one of them.

### *General Rules*

1. Every record in a Consecutive file has a relative record number. This relative record number specifies the logical ordinal position of the record in the file. The first record has relative record number 1, while subsequent records have relative record numbers of 2, 3, 4, and so on. Nonsequential access of a sequential file is a Wang extension to the Standard.
2. If the ACCESS MODE is SEQUENTIAL, in a Sequential file, records are accessed in the order in which they were written into the file; in a Relative file, records are accessed

in order of ascending relative record numbers of the existing records in the file; in an Indexed file, records are accessed in order of ascending prime record values.

3. If the ACCESS MODE is RANDOM, the record whose relative record number is in data-name-3 is the one that is accessed.
4. If the ACCESS MODE is DYNAMIC, records can be accessed sequentially or randomly.
5. If the workstation screen is being controlled by the Workstation I-O method, data name-3 identifies the row at which a READ or REWRITE statement begins. If an attempt is made to write a record that extends beyond the maximum size of the workstation file, it causes a runtime error.
6. If the file is external, data-name-3 in the RELATIVE KEY phrase in each associated FILE-CONTROL entry must reference the same external data item.

#### ALTERNATE RECORD KEY Clause

The ALTERNATE RECORD KEY clause specifies a nonprimary key that can be used to access a record in an Indexed file.

#### *Syntax Rules*

1. Data-name-4 must be an alphanumeric data item defined in a record description entry of the associated file. It cannot reference a group item that contains a variable occurrence data item. It can be qualified.
2. If ANSI compatibility is desired, data-name-4 must not be an item whose leftmost character position corresponds to the leftmost character position of either the prime record key or of any other alternate record key associated with this file. However, Wang COBOL 85 does allow record key overlap as an extension.
3. If ANSI compatibility is desired, data-name-4 must be entirely contained within the minimum record size specified for the file. Wang COBOL 85, however, allows an alternate record key to be positioned beyond the minimum record size as an extension.
4. Integer-4 is a Wang extension.

#### *General Rules*

1. An ALTERNATE RECORD KEY is a data item that is not the prime record key, but it can be used to access records.
2. The data description of data-name-4 as well as its relative location within the record must be the same as when the file was created.
3. If the Compiler option COMPAT74 is NO and integer-4 is not specified, the number of alternate record keys for the file must be the same as when the file was created. In addition, the alternate key paths for a record are determined by the record keys

associated with that specific record. It is not necessary to include all alternate record keys since a given key can appear in some records but not in others.

4. The **DUPLICATES** phrase specifies that an alternate key can have the same value in two or more records. If **DUPLICATES** is not specified, the value of an alternate key must be different for each record.
5. If the Compiler option **COMPAT74** is **NO** and integer-4 is not specified, the character positions comprising data-name-4 in any one record description entry are considered the same for all other record description entries for that file.
6. If the file has more than one record description entry, data-name-4 need be described only in one of them.
7. The combined lengths of the primary record key and the longest alternate record key cannot exceed 256 characters on the VS and 255 characters on the COBOL ReSource.
8. Integer-4 must match the ordinal placement of the alternate key indicated by data-name-4 at the time of file creation.
9. When integer-4 is specified, a **START** statement without the **KEY** phrase behaves as if a **KEY** phrase is present and specifies the file's prime record key.

### **FILE STATUS Clause**

The **FILE STATUS** clause designates a data item that indicates the success or reason for failure of an input-output operation.

#### *Syntax Rule*

Data-name-2 specifies the file status item. It must be a 2-character, alphanumeric data item that is not in the File section. It can be qualified.

#### *General Rules*

1. If the **FILE STATUS** clause is specified, each time an input-output operation is executed for the file a value is put into the file status item. Appendix E lists these values and their meanings.
2. When the specified file is a workstation file, the file status item indicates the key that terminated the input. This can be either the Enter key or one of the program function (PF) keys. If the Enter key is used, an at sign (@) is put in the rightmost character position. If a PF key is used, an uppercase or lowercase letter is put in this position. Appendix E lists these values and their meanings.

### **CURSOR POSITION Clause**

The **CURSOR POSITION** clause designates a data item that is updated by the system to reflect the current row and column position of the screen cursor.

### *Syntax Rules*

1. This clause can be used only with a DISPLAY file.
2. Data-name-4 must be a 01-level item with two halfword USAGE IS BINARY data items subordinate to it.

### **PFKEY Clause**

The PFKEY clause designates a data item that will receive the numeric value of the selected PF key following the execution of a READ or DISPLAY AND READ statement.

### *Syntax Rules*

1. This clause can be used only with a DISPLAY file.
2. Data-name-5 must be a numeric data item two digits long with a usage of DISPLAY.

### *General Rule*

Data-name-5 can receive values in the range of 00 to 32. A value of 00 indicates the Enter key, and 01 through 32 specify the corresponding PF keys (PF keys 17 to 32 are the shifted PF keys 1 to 16).

### **PADDING CHARACTER Clause**

The PADDING CHARACTER clause has no effect on the program. It, and the following rules, are included for ANSI compatibility.

### *Syntax Rules*

1. Literal-1 must be a nonnumeric literal, one character long.
2. Data-name-6 must be a 1-character alphanumeric data item and must not be defined in the File section. It can be qualified.

### **RECORD DELIMITER Clause**

The RECORD DELIMITER clause specifies how the length of a variable-length record on external medium is determined.

### *Syntax Rules*

1. The RECORD DELIMITER clause can be specified only for variablelength records.
2. The RECORD DELIMITER clause can be used only with a TAPE file.

## General Rules

1. The RECORD DELIMITER clause is used to indicate the way that the length of a variable-length record is determined on the external medium. The method used is reflected in the record area or the record size used within the program.
2. If the STANDARD-1 phrase is specified, the method is the one specified in American National Standard X3.27-1978, Magnetic Tape Labels and File Structure for Information Interchange, and International Standard 1001 1979, Magnetic Tape Labels and File Structure for Information Interchange.
3. Use of a data-name-7 is equivalent to using STANDARD-1. Data-name-7 can be any name, and it need not be defined anywhere else in the program.

## I-O-CONTROL Paragraph

The I-O-CONTROL paragraph designates a memory area that is shared among different files.

### Format

I-O-CONTROL.

[RERUN [ON file-name-1] EVERY integer-1 RECORDS OF file-name-2]

[SAME [ RECORD  
SORT  
SORT-MERGE ] AREA FOR file-name-3 {file-name-4}...]... .

### Syntax Rules

1. The two clauses can appear in any order.
2. The RERUN clause is scheduled for deletion from the next revision of the COBOL Standard. Wang COBOL 85 treats it as a comment.
3. If a sort or merge file is in a SAME clause it must contain one of the RECORD, SORT, or SORT-MERGE phrases.
4. The files referenced by file-name-3, file-name-4, etc., need not have the same organization or access. They cannot reference external files, however.
5. A file-name cannot appear in more than one SAME AREA clause.
6. SORT and SORT-MERGE are equivalent.

## General Rules

1. The SAME AREA clause (without a RECORD, SORT, or SORT-MERGE phrase) is treated as a comment for all but induced files on the VS, where it indicates that the files share a buffer pool.
2. The SAME RECORD AREA clause specifies that two or more files will use the same memory area for processing of records. Since all of the specified files can be open at the same time, a record of any file in this clause is considered a record of each of the specified files that is open in output mode, as well as a logical record of input file that was most recently read. Since this is an implicit redefinition of the area, records are aligned by the leftmost character position.
3. The SAME SORT AREA and SAME SORT-MERGE AREA clauses are treated as comments.

# Chapter 5 Data Division

## Introduction

The Data Division contains descriptions of the data the program will use. It has three sections: the File section, the Working-Storage section, and the Linkage section.

If the SYNCHRONIZED clause is specified on the Data Division header, all binary items and groups containing binary items (described in the Data Division) will contain the SYNCHRONIZED attribute. This occurs whether or not the clause is specified individually.

The File section contains definitions of the structure of the files used in the program. The Working-Storage section describes data items that are not part of files and their optional initialization values. The Linkage section is used mostly in called programs; it describes data that is made available by the calling program to the called program.

The general format for the Data Division is shown as follows.

### ***Format***

```
[ DATA DIVISION [ SYNCHRONIZED ] .  
    [ FILE SECTION . ]  
    [ WORKING-STORAGE SECTION . ]  
    [ LINKAGE SECTION . ] ]
```

## Data Description Entry

The data description entry specifies the characteristics of a data item in terms of its appearance on the printed page and some storage attributes.

### Format 1

```

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

[REDEFINES data-name-2]

[PICTURE IS character-string]

[ USAGE IS { BINARY
             COMPUTATIONAL
             COMP
             DISPLAY
             DISPLAY-WS
             INDEX
             PACKED-DECIMAL
             POINTER } ]

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

[ OCCURS [integer-1 TO] integer-2 TIMES
  [ DEPENDING ON data-name-3 ]
  [ { ASCENDING
      DESCENDING } KEY IS {data-name-4}... ]... ]
  [ INDEXED BY {index-name-1}... ] ]

[ [ NOT ] { SYNCHRONIZED
            SYNCHRONIZED } [ LEFT
                               RIGHT ] ]

[ { JUSTIFIED
    JUST } RIGHT ]

[ BLANK WHEN ZERO ]

[ IS EXTERNAL ]

```



[ IS GLOBAL ]  

$$\left[ \underline{\text{VALUE}} \text{ IS } \left\{ \begin{array}{l} \text{literal-1} \\ \text{user-figurative-constant} \end{array} \right\} \right] .$$

### **Format 2**

66 data-name-1 RENAMES data-name-2 [THROUGH data-name-3] .

### **Format 3**

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

### *Syntax Rules*

1. In format 1, a level-number can be from 01 through 49 or level number 77. Level numbers from 1 to 9 can be written with or without a leading zero.
2. If data-name-1 or the word FILLER is present, it must immediately follow the level-number.
3. The REDEFINES clause must immediately follow data-name-1 or the word FILLER. If neither is present, the REDEFINES clause must immediately follow the level-number.
4. The remaining clauses can be in any order.
5. Every elementary item, except an index data item, a binary data item, and the subject of the RENAMES clause must have a PICTURE clause. The subject of a RENAMES clause cannot have a PICTURE clause; for index and binary data items the PICTURE clause is optional.

### *General Rules*

### *All Formats*

1. The PICTURE, JUSTIFIED, and BLANK WHEN ZERO clauses can be specified only for elementary items.

### *Format 3*

2. Level 88 defines a condition-name and specifies the value, values, range, or ranges of values associated with it. Each condition-name requires a separate entry.
3. The condition-name entries for a particular conditional variable must immediately follow the entry describing the conditional variable. Any data description entry can have a condition-name except another condition-name; a level 66 data item; an index data item; or a group item containing items that are justified, synchronized, or have a usage other than DISPLAY.

## Data Description Clauses

The following data description entry clauses are presented in alphabetical order in the next sections.

### BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause suppresses printing or displaying of an item when its value is zero.

#### *Syntax Rule*

Only numeric or numeric-edited items with a usage of DISPLAY can have this clause.

#### *General Rules*

1. When an item described with this clause has the value zero, it contains nothing but spaces.
2. If an item is numeric, specifying this clause causes that item to be treated as numeric-edited
3. If the asterisk (\*) is used as a zero-suppression symbol in the PICTURE character-string, BLANK WHEN ZERO cannot be specified, and vice versa.

### Data-Name-FILLER Clause

A data-name gives a name to a data item. This name can be used to reference the data item in the program. If the data item is not explicitly referenced in the program, the word FILLER can be used instead.

#### *Syntax Rule*

If either data-name-1 or FILLER is used in a data description in the File, Working-Storage, or Linkage section, it must be the first word following the levelnumber in the data description entry.

#### *General Rules*

1. If this clause is omitted, FILLER is assumed.
2. If FILLER is used, the item cannot be explicitly referenced. However, a conditional variable can be a FILLER item because the variable need not be explicitly referenced, only the condition-name.

### EXTERNAL Clause

The EXTERNAL clause specifies a record as external. An external record and its constituent data items are available to every program in the run unit that describes them.

#### *Syntax Rules*

1. The EXTERNAL clause can be specified only for a level 0 1 data item in Working-Storage that has a data-name.
2. In the same program, all external data-names must be unique and cannot be the same as a system routine.
3. The EXTERNAL clause and the REDEFINES clause are mutually exclusive in a given data description entry.
4. The Standard prohibits a VALUE clause in any data description that contains an EXTERNAL clause or is subordinate to an entry that contains an EXTERNAL clause, except in a condition-name entry. As an extension, COBOL 85 allows a VALUE clause for these items. However, all data descriptions that describe the external item must contain the identical VALUE clause.

### *General Rules*

1. An external record can be accessed by any program in the run unit that describes or redefines it.
2. If two or more programs within a run unit describe the same external data record, each record-name must be identical, and the records must contain the same number of character positions.
3. An external data item's name is not necessarily global.

## **GLOBAL Clause**

The GLOBAL clause declares a data-name as global. A global data item is available to the program that declares it and to any other program that is directly or indirectly contained in the declaring program. Refer to the section titled "Scope of Names and Uniqueness of Reference" in Chapter 2 for additional information.

### *Syntax Rule*

The GLOBAL clause can be specified only for a level 01 item in the File section or the Working-Storage section that is not a FILLER item.

### *General Rules*

1. A data-name with the GLOBAL clause is a global name. All data-names subordinate to a global name, as well as all associated condition-names, are also global names.
2. If the GLOBAL clause is specified for an item with a REDEFINES clause, only the subject of the REDEFINES clause is global.

## **JUSTIFIED Clause**

The JUSTIFIED clause permits alternate positioning of data within a receiving data item.

### *Syntax Rule*

The JUSTIFIED clause can be specified only for elementary alphabetic or alphanumeric items.

### *General Rule*

When a receiving item is justified, and if the sending item is larger, the leftmost characters are truncated. If the sending item is smaller, the data is aligned at the rightmost character position with space fill on the left.

## **Level-Number Clause**

The level-number clause indicates the position of a data item within the structure of a record. In addition, it identifies non-contiguous data items and condition names.

### *Syntax Rules*

1. The first element in a data description entry must be a level-number.
2. Data description entries that are in file description or sort-merge file description entries can have level-numbers from 01 through 49, 66, or 88.
3. Entries in the Working-Storage and Linkage sections can have level-numbers 01 through 49, 66, 77, or 88.

## *General Rules*

1. Level-number 01 identifies the first and most inclusive (or only) data item in a record description.
2. Level 66 identifies RENAMEs entries.
3. Level 77 identifies noncontiguous working-storage and linkage data items.
4. Level 88 defines a condition-name.

## **OCCURS Clause**

The OCCURS clause eliminates having to write separate entries for table elements and provides information required for the application of subscripts.

## *Syntax Rules*

1. A data item that has a level-number of 01, 66, 77, or 88 or has a subordinate data item that is variable-occurrence cannot have an OCCURS clause.
2. Data-name-3 and data-name-4 can be qualified.
3. If the data item, or one of its subordinate data items, is to be referenced by indexing, the INDEXED BY phrase is required.
4. Index-name-1 must be a unique name within the program and cannot be defined or described elsewhere in the program.
5. The first data-name-4 must be the name of either the entry containing the OCCURS clause or an entry subordinate to the entry containing the OCCURS clause. Subsequent instances of data-name-4 must be subordinate to the entry containing the OCCURS clause.
6. When writing the OCCURS clause, data-name-4 must be written without the subscripting that would normally be required.
7. When both integer-1 and integer-2 are used, integer-1 must be zero or greater, and integer-2 must be greater than integer-1.
8. Data-name-3 must be an integer.
9. Data item-3 must not be subordinate to the entry containing the OCCURS clause.
10. If the OCCURS clause is specified for an external data item, data-name-3 must be an EXTERNAL data item that is described in the same Data Division.
11. A data description entry with a DEPENDING ON clause can be followed, within its record description, only by data description entries that are subordinate to it.
12. The Integer-1 entry is specified; the DEPENDING ON clause is required. Entries that specify the DEPENDING ON clause are variable-occurrence entries.
13. Data item-4 can contain an OCCURS clause only when it is the subject of the entry.
14. An entry that contains an OCCURS clause cannot be named in the KEY phrase of a FILE-CONTROL entry.

### General Rules

1. The OCCURS clause defines a table. When used in the Procedure Division, a data-name that contains an OCCURS clause must be either subscripted or indexed (except in a SEARCH statement). If an item with the OCCURS clause is a group item, all subordinate data-names must also be subscripted or indexed whenever used.
2. Except for the OCCURS clause itself, all data description clauses associated with an item that has an OCCURS clause apply to each occurrence of the item.
3. If the DEPENDING ON clause is not specified, the number of valid occurrences of the data item is specified by integer-2.
4. For a variable occurrence data item, the value of integer-2 represents the maximum number of valid occurrences, and the value of integer-1 represents the minimum number of valid occurrences.

When a variable occurrence data item (or any data item subordinate or superordinate to it) is referenced, the current value of data item-3 must be in the range of integer-1 to integer-2. An item in the table whose occurrence number exceeds the value of data item-3 does not participate in the operation.

5. If a group item that has a subordinate entry with a DEPENDING ON clause is referenced, the part of the group item used in the operation is determined as follows:
  - If data name is not part of the group item, that part of the table area indicated by the value of data-name-3 at the start of the operation is used
  - If data-name-3 is included in the group item and the group item is a sending item, that part of the table specified by the value of data-name-3 at the start of the operation is used in the operation. If the group item is a receiving item, all the elements of the group (as specified by integer-2) are used.
6. If a DEPENDING ON clause is specified in a record description entry, and the associated file description or sort-merge description entry contains the VARYING phrase of the RECORD clause, the records are variable-length. If the DEPENDING ON phrase of the RECORD clause is not specified before the execution of any RELEASE, REWRITE, or WRITE statement, data-name-3 of the OCCURS clause must be set to the number of occurrences to be written.
7. If the KEY phrase is used, the table elements must be stored in ascending or descending order according to the values of data-name-4. If more than one data-name-4 is specified, the first one specified is the most significant, the one to its right is the next most significant, and so on.

### PICTURE Clause

The PICTURE clause describes the general characteristics and editing requirements of an elementary data item.

### Syntax Rules

1. The PICTURE character-string consists of up to 30 characters. The allowable combinations of characters, as discussed in the general rules, determine the category of the elementary item.
2. The PICTURE clause can be specified only for an elementary item.
3. Every elementary item except a data item whose usage is INDEX, BINARY or POINTER, and the subject of a RENAME clause must have a PICTURE clause. For items, whose usage is INDEX or BINARY, this clause is optional.
4. The PICTURE of an index data-item can contain *only* the characters S and 9.

5. The lowercase letters a, b, p, s, v, x, and letter pairs cr, and db are equivalent to the corresponding uppercase letters and letter pairs when used in a PICTURE character-string. All other lowercase letters are not equivalent to their corresponding uppercase letter.
6. The asterisk cannot be used as a zero suppression symbol if the entry has a BLANK WHEN ZERO clause.

### General Rules

1. An integer in parentheses in a PICTURE character-string is called a repetition factor and indicates the number of times the character to its left is repeated. Thus X(5) and XXXXX are equivalent.
2. The rules governing the use of the PICTURE clause are divided into five areas: data categories, size of the data item, symbol functions, editing rules, and precedence rules.

### Categories of Pictured Data

The following five categories of data can be described with a PICTURE clause:

**Alphabetic Data** - An item is alphabetic if its character-string contains only the symbol A and, optionally, a repetition factor. An alphabetic item contains alphabetic characters or spaces.

*Note: Wang COBOL allows the characters '0' through '9' in VALUE clauses of data items defined as alphabetic.*

**Alphanumeric Data** - An item is alphanumeric if its character-string contains certain combinations of the symbols A, X, and 9. (A characterstring containing all A's or all 9s does not define an alphanumeric item.) An alphanumeric item can contain any characters in the computer character set. Alphanumeric items are treated as if the character-string contained all X's.

**Alphanumeric Edited Data** - An item is alphanumeric edited if its character-string consists of certain combinations of the A, X, 9, B, 0, and slash (/) symbols, and it contains at least one A or X, as well as at least one B or 0 or slash (/).

An alphanumeric edited item can contain any characters in the computer character set.

**Numeric Data** - An item is numeric if its character-string contains only the symbols 9, P, S, and V. When unsigned, the contents of a numeric item must be one or more numeric characters. If the item is signed, it can also contain a plus sign (+), a minus sign (-), or another representation of an operational sign. The number of digit positions described by the character-string must be from 1 to 18.

**Numeric Edited Data** - An item is numeric edited if at least one character in its character-string is a B, slash (/), Z, 0, comma (,), period (.), asterisk (\*), plus sign (+), minus sign (-), CR, DB, or the currency symbol. In addition, 9s, P's or a V can appear in the string. The number of digit positions described by the character-string must be from 1 to 18.

The content of each of the character positions must be consistent with the corresponding PICTURE symbol.

### Size of Elementary Item

The size of an elementary item is the number of character positions (bytes) occupied by the item. The size of an item is determined by the item's PICTURE and USAGE clauses where the PICTURE clause determines the size of the data on the printed page, and the USAGE clause may affect its internal storage size. The affect of picture-string symbols on the size of a data item are indicated in the following section.

## Picture-String Symbols

*Note: The symbols A, B, P, X, Z, 0, 9, comma (,), asterisk (\*), slash (/), plus sign (+), minus sign (-), and the currency symbol (\$), can appear more than once in a PICTURE string. The symbols S, V, CR, DB, and period (.) can appear only once in a PICTURE string.*

The functions of the various symbols in the PICTURE character-string are

**A** - Each A represents a character position that can contain an alphabetic character or, as a Wang extension, a digit. Each A is counted in the size of the item.

**B** - Each B represents a character position that will receive a space (blank) when the item is printed or displayed. Each B is counted in the size of the item.

**P** - Each P indicates a decimal scaling position. This position is not counted in the size of the item. However, it is counted in determining the number of digits in numeric and numeric-edited items.

All P's in the PICTURE must be contiguous and must be either the leftmost or the rightmost characters in the string. If they are the leftmost characters of the string, a decimal point is assumed to be to their left; if they are the rightmost characters, a decimal point is assumed to be to their right. Because of this assumed decimal point, the symbol V is redundant when P's are used. The P and the period cannot both appear in a PICTURE characterstring.

When a data item has a P in its picture, certain operations using that item result in the algebraic value of the data item being used in the operation instead of the character representation. This algebraic value assumes that the decimal point is located as described previously and that there is a zero in each position designated by a P. These operations are

- Any operation requiring a numeric operand
- A MOVE statement in which the sending operand is numeric.
- A MOVE statement in which the sending operand is numeric-edited and the receiving operand is either numeric or numeric-edited
- A comparison operation in which both operands are numeric.
- In all other operations, the digit positions specified by a P are ignored and are not counted in the size of the operand

**S** - An S indicates an operational sign. The S must be the leftmost symbol in the string. If the entry has a SIGN clause with the SEPARATE CHARACTER phrase, the S is counted in determining the size of the elementary item; otherwise, it is not counted.

**V** - A V indicates the location of an assumed decimal point. There can be only one V in the character-string. Since the V does not represent a character position, it is not counted in determining the size of the item. A V is redundant when it is the rightmost symbol in a string.

**X** - Each X represents a character position that can contain any character from the computer character set. Each X is counted in the size of the item.

**Z** - Each Z represents a character position that is filled with a space if it contains a leading zero. All Z's must be contiguous and must represent the leftmost digit positions of the string. Each Z is counted in the size of the item.

**9** - Each 9 in a character-string represents a digit position containing a numeric character and is counted in the size of the item.

**0** - Each 0 (zero) in a character-string represents a character position into which the character zero is inserted and is counted in the size of the item.

/ - Each slash character in a character-string represents a character position into which the slash character is inserted and is counted in the size of the item.

, - Each comma in a character-string represents a character position into which the comma character is inserted, and is counted in the size of the item.

. - The period represents an actual decimal point. This position is counted in the size of the item.

*Note: If the DECIMAL-POINT IS COMMA clause is included in the SPECIAL-NAMES paragraph, the functions of the period and comma are interchanged. Consequently, the rules for the period apply to the comma and the rules for the comma apply to the period.*

+ - CR DB - These four symbols are the editing sign control symbols. They are mutually exclusive in a PICTURE string. Since each character is counted in determining the size of the data item, CR and DB count as two characters.

\* - Each asterisk in the character-string represents a character position into which an asterisk is placed when the content of that position is zero. All asterisks must be contiguous and represent the leftmost digit position(s) of the string. Each asterisk is counted in the size of the item.

cs - The currency symbol (cs) represents a character position into which a currency symbol is placed. This symbol is represented in a character-string by either the currency sign (\$) or by any single character specified in the CURRENCY SIGN clause of the SPECIAL-NAMES paragraph. This symbol is counted in determining the size of the item.

## Editing

Editing of an item can be done by either insertion or by suppression and replacement. The four types of insertion editing are

- Simple insertion
- Special insertion
- Fixed insertion
- Floating insertion

The two types of suppression and replacement editing are

- Zero suppression and replacement with spaces
- Zero suppression and replacement with asterisks

The type of editing that an item can have depends on its category, as shown in the following table.

<b><u>Category</u></b>	<b><u>Type of Editing</u></b>
-Alphabetic, numeric, alphanumeric	None
-Alphanumeric edited	Simple insertion
Numeric edited	All



## Editing Rules

The following editing rules apply:

1. Simple insertion editing results in the insertion character occupying the same character position in the edited item as it occupies in the PICTURE character-string.

The B, 0, comma (,), and slash (/) are simple insertion characters. If the comma is the last symbol in the character-string, the PICTURE clause must be the last clause of the data description entry and must be immediately followed by a period.

2. Special insertion editing results in the insertion character occupying the same character position in the edited item as it occupies in the PICTURE character-string.

The period (.) is the special insertion character. In addition, it also represents the decimal point for alignment purposes. The assumed decimal point symbol (V) and the period cannot both be used in the same characterstring. If the period is the last symbol in the PICTURE character-string, the PICTURE clause must be the last clause of that data description entry and must immediately be followed by a separator period.

3. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupies in the PICTURE character-string.

The currency symbol and the editing sign control symbols (+ - CR DB), perform fixed insertion editing. There can be at most one currency symbol and one of the editing sign control symbols in a PICTURE string.

When CR or DB is used, it must be the rightmost character position. The uppercase letters are the characters that are inserted.

The + and - symbols must be either the leftmost or rightmost character positions that are counted in the size of the item.

The currency symbol must be the leftmost character position that is counted in the size of the item, except that it can be preceded by a + or - .

Editing sign control symbols produce the results shown in the following table.

Editing Symbol	Printed Result	
	Data Item Is Positive or Zero	Data Item Is Negative
+	+	-
-	One space	-
CR	Two spaces	CR
DB	Two spaces	DB

4. Floating insertion editing uses the currency symbol and editing sign control symbols (+, -). These symbols are mutually exclusive in a given PICTURE character-string.

Floating insertion editing is indicated by using a string of at least two identical floating insertion editing characters. Any of the simple insertion characters can be embedded in this string or can be to the immediate right of it. When this is so, these simple insertion characters are considered part of the floating string. When the currency symbol is the floating insertion character, CR or DB can be to the immediate right of the string.

The second floating character from the left represents the leftmost limit of the numeric data that can be stored in the data item. All the characters at or to the right of this character can contain nonzero numeric data.

Floating insertion editing is indicated in one of two ways. In the first method, any or all of the leading numeric character positions to the left of the decimal point are represented by the insertion character. In the second method, all of the numeric character positions in the item are represented by the insertion character.

Having the insertion characters only to the left of the decimal point results in a single insertion character being put into the character position immediately preceding either the decimal point or the first nonzero digit in the data, whichever is further to the left. The character positions preceding the insertion character are replaced with spaces.

When all the character positions are floating insertion characters, at least one of the insertion characters must be to the left of the decimal point. The result depends upon the value of the data. If the value is zero, the entire data item contains spaces. If the value is not zero, the result is the same as if insertion characters were used only to the left of the decimal point.

The character inserted into a PICTURE character-string when + or - is used as a floating insertion character depends upon the value of the data item, as shown in the following table.

<b>Editing Symbol</b>	<b>Printed Results</b>	
	<b>Data Item Is Positive or Zero</b>	<b>Data Item Is Negative</b>
<b>+</b>	<b>+</b>	<b>-</b>
<b>-</b>	<b>One space</b>	<b>-</b>

If the receiving data item is not large enough to contain the sending item plus the number of nonfloating insertion characters put into it, plus one floating insertion character, truncation occurs. In this case, the value of the data used for editing is the value after truncation.

5. Zero suppression and replacement editing use the characters Z and \* to suppress leading zeros. These two symbols are mutually exclusive in a given PICTURE character-string. When the suppression character is Z, leading zeros are replaced by spaces. When the suppression character is an asterisk, leading zeros are replaced by asterisks.

There are two ways of doing zero suppression and replacement. The first method is by representing one or more of the leading numeric character positions to the left of the decimal point by suppression symbols (Z and \*). In this case, leading zeros in the item are replaced by the replacement character. This replacement ceases at the first nonzero digit in the item or at the decimal point, whichever comes first.

The second method is by representing all of the numeric character positions in the item by suppression symbols. In this case, if the value of the item is not zero, the result is the same as shown in the preceding paragraph. However, when the value of the item is zero and Z is used as the suppression symbol, all positions in the item, including any editing characters, contain spaces; if the asterisk is used, all positions in the item except the decimal point contain asterisks.

6. Floating insertion editing and editing by zero suppression and replacement are mutually exclusive in a PICTURE clause.
7. The +, -, \*, Z, and the currency symbol are mutually exclusive when used as floating replacement characters in a given character-string.

8. If the asterisk (\*) is used as a replacement character, BLANK WHEN ZERO cannot be specified for the data-item, and vice-versa.

## REDEFINES Clause

The REDEFINES clause permits an area in main storage to have different data descriptions.

### *Syntax Rules*

1. The REDEFINES clause must immediately follow the subject of the entry.
2. The level-number of data-name-2 must be identical to the level-number of the subject of the entry, but it cannot be level 66 or level 88.
3. This clause cannot be used in a level 01 entry in the File section.
4. Neither data-name-1 nor data-name-2 can include a variable occurrence data item.
5. Data-name-2 cannot be qualified or subscripted.
6. As a Wang extension, data-name-2 can have an OCCURS clause in its description.
7. If data-name-2 is external or has a level-number other than 01, it must have at least as many character positions as data-name-1.
8. Multiple redefinitions of the same character positions are permitted; however, they must all refer to the same data-name-2.
9. Entries that give new descriptions of character positions cannot contain a VALUE clause, except for condition-name entries.
10. New character position entries cannot appear between the entry defining the area of data-name-2 and the new description entries. A numerically lower level-number cannot appear between the data description entries of data-name-2 and the subject of the entry (data-name-1 or FILLER).
11. Data-name-2 can be subordinate to an entry containing a REDEFINES clause.

### *General Rules*

1. Storage allocation starts at data-name-2 and continues over a storage area sufficient to contain the number of character positions in data item-1 (or the FILLER item).
2. When more than one data description specifies a character position, the data-name from any of those descriptions can be used to reference that character position.

## RENAMES Clause

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

### *Syntax Rules*

1. A record can have any number of RENAMES clauses.
2. All RENAMES entries referring to data items within a given logical record must immediately follow the last data description entry of that record.
3. Data-name-1 cannot be used as a qualifier and can itself be qualified only by the names of the associated level 01, FD, or SD entries. Neither data-name-2 nor data-name-3 can have an OCCURS clause in its data description nor can either be subordinate to an item that has an OCCURS clause in its data description entry.
4. Data-name-2 and data-name-3 must be names of elementary items or groups of elementary items in the same logical record and cannot be the same data-name. A level 66 entry cannot rename another level 66 entry, nor can it rename a level 77, 88, or 0 1 entry.
5. Data-name-2 and data-name-3 can be qualified.
6. None of the items within the range from data-name-2 to data-name-3 can be a variable occurrence data item.
7. The beginning of data item-3 must not be to the left of the beginning of data item-2. The end of data item-3 must be to the right of the end of data item-2. Data-name-3 cannot, therefore, be subordinate to data-name-2.
8. The level number 66 must begin in area A.

### *General Rules*

1. When data-name-3 is not specified, all of the data attributes of data-name-2 become the data attributes for data-name-1.
2. When data-name-3 is specified, data-name-1 is a group item that includes all elementary items from data-name-2 (or the first elementary item in data-name-2 if data-name-2 is a group item) to data-name-3 (or the last element in data-name-3, if data-name-3 is a group item).

## SIGN Cause

The SIGN clause specifies the position and mode of representation of an operational sign.

### *Syntax Rules*

1. The SIGN clause can be specified for a numeric data item whose PICTURE contains an S or for a group item that contains at least one such numeric data item.
2. The data item must have a usage of DISPLAY, explicitly or implicitly.
3. If the file containing the item has a CODE-SET phrase, the item must have the SEPARATE CHARACTER phrase.

### *General Rules*

1. The SIGN clause specifies the position and mode of representation of the operational sign for an elementary item or, if applied to a group item, for each numeric data item subordinate to the group. For a group item, this clause applies only to those constituent items whose PICTURE contains an S.

2. If an item that has a SIGN clause has a subordinate item with a SIGN clause, the subordinate item's SIGN clause takes precedence for that subordinate item.
3. A data description entry that does not contain a SIGN clause but has the character S in its PICTURE character-string has a TRAILING SEPARATE sign. General Rules 4 through 6 do not apply to such items.
4. If SEPARATE is not specified, the S in the PICTURE string is not counted in determining the size of the item.
5. If SEPARATE is specified, the operational sign is in the leading (or trailing) character position of the item. This character position is not a digit position. The operational signs are the characters ( + and - ). The S in the PICTURE string is counted in determining the size of the item.
6. If an item's PICTURE string contains an S and the item has a SIGN clause, any conversion necessary for computation or comparisons takes place automatically.

## SYNCHRONIZED Clause

The SYNCHRONIZED clause causes an elementary item to be aligned on a byte, halfword, fullword, or doubleword boundary in memory.

### *Syntax Rule*

This clause can be specified for any item that is not an INDEX data item.

### *General Rules*

1. If SYNCHRONIZED is specified for an elementary item whose usage is other than BINARY, it is treated as a comment.
2. If SYNCHRONIZED is specified for a halfword binary item, that item is aligned to the next highest halfword boundary. If SYNCHRONIZED is specified for a fullword or doubleword binary item, that item is aligned to the next highest fullword boundary. Because of this, unused bytes may occur between the previous item and this item.
3. The default for a binary item is not synchronized. That default can be explicitly requested with the NOT SYNCHRONIZED clause.
4. If a binary item is not synchronized, that item is aligned to the next byte boundary. No unused bytes exist between the previous item and this item.
5. If a synchronized binary item is subordinate to a group item, the length of the group item includes any character positions that were left unused as a result of synchronizing that binary item.
6. If the SYNCHRONIZED clause is specified for a group item, it does not apply to the group item itself, but rather to all the eligible subordinate elementary data items in the group. When the SYNCHRONIZED clause is specified both for a group item and for a subordinate item, the clause at the subordinate level takes precedence.
7. If the SYNCHRONIZED clause is specified for an item that has an OCCURS clause, or for an item subordinate to an item that has an OCCURS clause, each occurrence of that table item is synchronized, and any resulting FILLER is generated for each occurrence of the table item.
8. If a group item contains synchronized binary data items that are aligned on a fullword boundary, the group item is also aligned on a fullword boundary. If the group item contains synchronized binary data items that are aligned on a halfword boundary, the group item itself is aligned on a halfword boundary. In all other cases, the group item is not synchronized and begins with the next available byte.

9. Records in the File and Working-Storage sections always begin on a fullword boundary.
10. LEFT and RIGHT are treated as comments.
11. If an item containing a SYNCHRONIZED clause has a subordinate item with a NOT SYNCHRONIZED clause, the subordinate item's clause takes precedence for that subordinate item.

## USAGE Clause

The USAGE clause specifies the format of a data item.

### *Syntax Rules*

1. The USAGE clause cannot be used for entries with level-number 66 or 88.
2. If this clause is specified for a group item and for a subordinate item, both entries must have the same usage.
3. Only elementary numeric items, or group items whose subordinate elementary items are numeric, can have a usage of BINARY, COMPUTATIONAL, or PACKED-DECIMAL.
4. An item with a usage of INDEX can be explicitly referenced only in a SEARCH or SET statement, a relation condition, or in the USING phrase of either a Procedure Division header or a CALL statement.
5. An item with a usage of INDEX cannot have a BLANK WHEN ZERO, JUSTIFIED, SYNCHRONIZED, or VALUE clause.
6. An item with a usage of INDEX cannot be a conditional variable; that is, it cannot have a level 88 entry associated with it.
7. As a Wang extension, an item with a usage of INDEX can have a PICTURE clause.
8. Only an elementary item without a PICTURE clause can have a usage of POINTER. A usage POINTER is a Wang extension.

### *General Rules*

1. If a group item has a USAGE clause, it applies to each elementary item in the group.
2. The USAGE clause determines the way an item is represented internally. BINARY items are represented in base 2; COMPUTATIONAL and PACKED-DECIMAL items are represented in base 10, packed two characters per byte; and DISPLAY items are represented in base 10 as ASCII characters.
3. Although the USAGE clause determines the way an item is stored internally, it generally has no effect on the use of the data item.

4. If a BINARY item has no PICTURE, it occupies two bytes. If it has a PICTURE, the amount of storage depends on the number of digits specified in the PICTURE. The following list explains this further:

**Digits in PICTURE    Storage Occupied**

1 through 4            2 bytes

5 through 9           4 bytes

10 through 18        8 bytes

5. When a binary item has a PICTURE specified, it need not be an integer. (Use of noninteger binary is not recommended.)
6. A usage of COMPUTATIONAL is the same as PACKED-DECIMAL. The item is stored as a decimal number.
7. A usage of DISPLAY specifies that the item is stored as ASCII characters. This is the default for an item with no USAGE clause.
8. Only a level 01 data item in working storage can have a usage of DISPLAY-WS. Such a record is automatically formatted for screen display, but since this does not occur until a DISPLAY AND READ statement is executed, data items in a display record cannot be accessed until then. For additional information on DISPLAY-WS records, refer to the *COBOL ReSource Programmer's Guide*.

When the USAGE clause is present, the compiler generates a 4-byte display order area for screen control. This area can be examined and modified with a MOVE statement.

The associated field attribute characters (FACs) of subordinate data items can also be accessed before a DISPLAY AND READ statement is executed, using MOVE statements.

9. An item with a usage of INDEX is called an index data item. Its value corresponds to an occurrence number of a table element.

If an INDEX data item does not have a PICTURE clause, it is stored the same way as a 4-byte BINARY item. However, if the Compiler option CONWAT74 is YES (refer to Appendix B), it is stored as a 2-byte item.

As an extension, a PICTURE clause can be specified with an index data item. The PICTURE can indicate up to nine digit positions. If the number of digit positions is four or fewer, the item is allocated two bytes of storage. If the number of digit positions is more than four, the item is allocated four bytes of storage.

If an INDEX data item is part of a group item that participates in a MOVE or input-output statement, no conversion from its value to an occurrence number takes place.

10. If an item has a usage of PACKED-DECIMAL, it is stored as 4-bit binary-coded-decimal digits, packed two to a byte, with the rightmost 4-bit positions reserved for the sign.

For example, both the number 50 and the number 501 require two bytes:

```
0000 0101 0000 1111    0101 0000 0001 1111
0    5    0    +       5    0    1    +
```

11. An item with a usage of **POINTER** is a pointer data item. The contents of a pointer data item are the address of some other data item. Because of this, a **POINTER** item can be used only in a conditional expression, the **SET** statement, and in the **USING** phrase of a **CALL** statement. If the contents of a pointer item are zero, it means the item does not currently indicate the address of another item.

## VALUE Clause

The **VALUE** clause indicates the initial value of items in working storage and specifies the values of condition-names.

### Syntax Rules

1. A nonnumeric literal in a **VALUE** clause cannot be greater than the size permitted by the item's **PICTURE** clause.
2. A numeric literal in a **VALUE** clause must be within the range of values permitted by the item's **PICTURE** clause.
3. If an item's **PICTURE** character-string indicates the item is signed, the numeric literal in the **VALUE** clause must be signed.
4. A data item that is in an external record can have a **VALUE** clause only for any associated condition-names.
5. An **INDEX** data item cannot have a **VALUE** clause.
6. In format 2, literal-1 must be less than literal-2. This form of **VALUE** clause can be used only with condition-names.

### General Rules

1. The **VALUE** clause must not conflict with other clauses in the data description of an item or in the data descriptions within the hierarchy of the item.

For numeric items, literals in a **VALUE** clause must be numeric. The literal is aligned in the data item according to the standard alignment rules.

For alphabetic, alphanumeric, alphanumeric-edited, and numeric-edited items, literals in a **VALUE** clause must be nonnumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric.

For an edited item, literals in the **VALUE** clause must be in edited form. That is, they must contain any insertion characters.

2. In the **File** and **Linkage** sections, the **VALUE** clause can be used only in condition-name entries.
3. In the **Working-Storage** section, the **VALUE** clause can both specify the initial value of a data item and any condition-names associated with that item.
4. The value in a **VALUE** clause is assigned to the item whenever the program is placed into its initial state. A data item that does not have a **VALUE** clause is not initialized.



5. A data item that has a REDEFINES clause or that is subordinate to an entry with a REDEFINES clause cannot have a VALUE clause, except in associated condition-name entries.
6. If a VALUE clause is used with a group item, subordinate items cannot have a JUSTIFIED, SYNCHRONIZED, VALUE, or USAGE clause (other than USAGE IS DISPLAY.) Also, the associated literal must be either nonnumeric or a figurative constant. The group area is initialized without consideration for the individual elementary or group items or any implicit FILLER contained within the group.
7. A data item is considered a variable occurrence item if it is itself a variable occurrence data item or is subordinate to a variable occurrence data item or if it is a group data item that contains a variable occurrence data item.

Variable occurrence data items can affect the initialization of associated data items that contain a VALUE clause. Initialization behaves as if the data item in the DEPENDING ON phrase of the item's OCCURS clause had been set to the maximum possible number of occurrences.

If a VALUE clause is used with a data item specified in a DEPENDING ON phrase, that value is placed in the data item after the variable occurrence data item is initialized.

8. If a VALUE clause is used with a data description that either contains an OCCURS clause or that is subordinate to an OCCURS clause, every occurrence of the associated data item is assigned the specified value.
9. A format 2 VALUE clause is required for condition-name entries. The VALUE clause and the condition-name itself are the only two clauses permitted in the entry.
10. Initialization of a data item with a VALUE clause is not affected by the presence of a BLANK WHEN ZERO or a JUSTIFIED clause.
11. A figurative constant can be used for a literal.

## Workstation Screen Description Entry

A workstation screen description entry specifies the format and contents of a screen that is to be displayed by a DISPLAY AND READ statement. Each screen is treated as one record of a workstation file defined in the File section. This entry allows the programmer to control data entry to, and data movement to and from, the workstation file.

### Format

```

01  record-name USAGE IS DISPLAY-WS

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

      [ COLUMN NUMBER IS integer-1 ]

      [ { ROW } NUMBER IS integer-2 ]
      [ { LINE } ]

      [ PICTURE IS character-string ]

      [ OBJECT IS data-name-2 ]

      [ SOURCE IS data-name-3 ]
      [ VALUE IS literal-1 ]

      [ OCCURS integer-3 TIMES ]

      [ RANGE IS { NEGATIVE
                   { POSITIVE
                   table-name
                   FROM { data-name-4 } TO { data-name-5 }
                   literal-2      literal-3 } } ] ]

```

### Syntax Rules

1. USAGE IS DISPLAY-WS is valid only in the Working-Storage section.
2. The record-name and USAGE IS DISPLAY-WS must be the first entries in the record description, followed by data-name-1 or FILLER. The other clauses can be in any order.
3. A group item must have a ROW clause and cannot have any other clause except the OCCURS clause.
4. If an elementary item has an OCCURS clause, it must also have a ROW and COLUMN clause.
5. If an item has an OCCURS clause, it cannot have a VALUE clause.
6. Data-name-1 must be unique in the program.
7. Data-names-2 through 5 can be qualified.

8. Data-name-4, data-name-5, and table-name cannot be external items.
9. The level-numbers can be from 02 to 49.

### ***General Rules***

1. If an item is a group item, it specifies a row or a group of rows.
2. If an item is an elementary item, it specifies a field within a row.
3. All elementary items in a group item are associated with the same row. Hence, they must not cause the row to overflow.
4. If group items are not used, fields within a row are accessed by elementary items.
5. Areas of the display screen that are not specified in the screen description are blanked.
6. A modifiable field is a field that has an OBJECT clause. Blanks in modifiable fields and all characters in uninitialized modifiable fields display as pseudo-blanks.
7. The PICTURE clause is the same as for normal data items.

## **COLUMN Clause**

The COLUMN clause specifies the beginning column position of the data item to be displayed.

### ***Syntax Rules***

1. The COLUMN clause must be specified for an elementary display item that has an OCCURS clause.
2. Integer-1 can be from 2 through 80. If 1 is specified, no field attribute character is generated.

### ***General Rules***

1. A nondisplaying control character, referred to as a field attribute character (FAC), occupies the column preceding the one designated by the COLUMN clause. In addition, there is a nominal FAC before each line's first column. The FAC controls the display attributes of its associated field. The attributes a field can have are low intensity, protected, and alphanumeric. The exact make-up of a FAC depends on the OBJECT clause of the associated field.
2. The COLUMN position of the end of the field to be displayed is one less than the sum of integer-1 and the number of characters specified for the field in the PICTURE clause.
3. If the ending position exceeds column 80, the field continues with column 1 of the next line, and the portion of the line that overlaps to the next line is controlled by the nominal FAC before that line's first column.
4. At least one space must separate one field from another. If the COLUMN clause is not specified, the field's beginning column is such that one space separates it from the ending column of the preceding field. If there is no preceding field and the COLUMN clause is not specified, the field begins at column 2 of line 1.

5. When a COLUMN clause is specified for an item that designates a field in a **row and** that item has an OCCURS clause, integer-1 indicates the starting column of the first field in the row. The starting column of each subsequent field is two more than the ending column of the previous field. Thus, each field is separated from the next field by one space.
6. When a COLUMN clause is specified for an item that designates a row and that item has an OCCURS clause, integer-1 indicates the starting column position of the specified field in the first row and in each subsequent row to which the OCCURS clause applies.

## ROW Clause

The ROW clause specifies the row (line) in which the data item is displayed.

### *Syntax Rules*

1. Integer-2 can be from 1 to 24.
2. LINE is equivalent to ROW.

### *General Rules*

1. If the ROW clause is omitted, the previous line is assumed. If no previous line was specified, line 1 is assumed.
2. The ROW clause must be specified for all group items and for elementary items that have an OCCURS clause.

## RANGE Clause

The RANGE clause specifies the valid limits of user input for a field. These limits are used by the DISPLAY AND READ statement to determine whether a modifiable field has a valid value or not.

### *Syntax Rules*

1. Data-name-4 and data-name-5 must be elementary data items, and cannot be external. Either can be qualified.
2. Literal-2 (or the contents of data-name-4) must be less than or equal to literal-3 (or the contents of data-name-5). The ASCII collating sequence is used to determine which alphanumeric character precedes another.

### *General Rules*

1. NEGATIVE is any value less than zero; POSITIVE is any value greater than zero.
2. Table- I must be a numeric or alphanumeric data item whose description contains an OCCURS clause. The table's PICTURE must be of the same length and type as that of the display data item. If the item is equal to the value of any table element it is valid.
3. If the item with the RANGE clause is alphanumeric or numeric-edited and the items specified in the RANGE clause are numeric, the data entered is converted to a numerical

value before being validated. If the RANGE clause items are alphanumeric, the validation is made according to the ASCII collating sequence.

## **SOURCE and VALUE Clauses**

The SOURCE clause specifies a data item from which a value is obtained for the initial contents of a screen data item. The VALUE clause specifies the initial contents of a screen data item.

### ***Syntax Rules***

1. The SOURCE clause and the VALUE clause are mutually exclusive for an item.
2. Data-name-3 cannot be subscripted or be the name of a DISPLAY-WS record or field. It can have an OCCURS clause and be qualified.
3. The SOURCE clause cannot be specified for a group item.

### ***General Rules***

1. The transfer of data from data-name-3 occurs at execution time in accordance with the rules of the MOVE statement.
2. If the SOURCE clause and an OCCURS clause apply to the same display item, data-name-3 must have the same size as the corresponding data-name in the OCCURS clause and be able to participate in a valid MOVE operation between the two.
3. The rules governing the VALUE clause for workstation screen description entries are the same as those for the VALUE clause in data description entries, except that in a DISPLAY-WS record (and only there), the VALUE clause can assign a numeric literal to an item with a nonnumeric picture.

## **OBJECT Clause**

The OBJECT clause specifies a data item into which data is moved from a field after validation. It also determines display attributes for fields.

### ***Syntax Rules***

1. The OBJECT clause cannot be specified for a group display item.
2. Data-name-2 cannot be subscripted or be a group item. It can contain an OCCURS clause and be qualified.

### ***General Rules***

1. The transfer of data to data-name-2 follows the rules of the MOVE statement. If data name-2 is alphanumeric, the move occurs without any conversion. If it is numeric, the move follows the rules for the MOVE WITH CONVERSION statement except
  - The value in the object field is not automatically aligned according to an implied decimal point or scaling in the source field. If the operator does enter an actual decimal point from the workstation, the data in the object field is aligned according

to it. Otherwise, the decimal point is assumed to be positioned to the right of the last digit.

- If the move is from a numeric-edited screen field, any currency sign, CR, DB, asterisk, slash, space (represented by the 'B' character symbol), and comma (or period if DECIMAL-POINT IS COMMA is specified) are removed
- 2. If the OBJECT clause is not specified, the FAC for the field is generated as low intensity, protected, alphanumeric. If the OBJECT clause is specified, the FAC is a function of the PICTURE clause of the elementary item. For numeric items, the FAC is high intensity, modifiable, numeric only. For alphanumeric items, the FAC is high intensity, modifiable, alphanumeric, uppercase.
- 3. If the OBJECT clause and an OCCURS clause apply to the same display item, data name-2 must have the same dimensions as the corresponding data item in the OCCURS clause and must also have a compatible PICTURE clause.

## **OCCURS Clause**

The OCCURS clause defines a table of display items that can appear as a repeating field, a row, or a group of rows on the screen.

### ***Syntax Rules***

1. The OCCURS clause can be used with both group and elementary items.
2. Up to three nestings of OCCURS clauses are permitted.

### ***General Rules***

1. When a group item has an OCCURS clause, it indicates that a row of fields or a group of rows is repeated. For an elementary item, it indicates that a field is repeated within a row.
2. When an elementary has an OCCURS clause, the starting column position of the first field in the row must be specified by the COLUMN clause. The starting position of each subsequent field is two more than the ending position of the previous field. Thus, each field is separated from the next by one blank space. However, a field cannot be repeated so many times as to cause a display beyond column 80.
3. When a group item has an OCCURS clause, placement of fields within the row (or group of rows) is determined by the COLUMN clauses of the elementary display items. The fields in the repeated row or group of rows are started in the same columns as they were in the preceding row (or in the relevant row within the preceding group). However, a row or group of rows cannot be repeated so many times as to cause a display beyond row 24.

## **Noncontiguous Data**

Data items that bear no hierarchical relationship to one another need not be grouped into records. Such items are called noncontiguous data items and have the special level-number 77. Optionally, such items can be defined simply as level 01 items that are not subdivided.

A data description entry for a noncontiguous elementary data item must have a level-number, a data-name, and at least a PICTURE or a USAGE clause to describe its storage. The other data description clauses are optional.

## **File Section**

The File section defines the files that the program uses, as well as the types of record layouts in each file. A file definition consists of a file description entry followed by a record description entry for each record type in the file. The VALUE clause is valid in the File section only in condition name (level 88) entries, since the initial value of a data item in the File section is set when a record is read or being prepared to be written.

## **File Description Entry**

A file description entry consists of a level indicator (FD), a file-name and scope. and a series of clauses that specify the

- Names of data records
- Presence or absence of label records
- Value of label items, logical and physical record sizes
- Number of lines to be printed on a logical printer page

## Format

FD file-name-1  
[ IS GLOBAL ]  
[ IS EXTERNAL ]  
[ BLOCK CONTAINS [integer-1 TO] integer-2 { RECORDS  
CHARACTERS } ]  
[ RECORD { CONTAINS [integer-3 TO] integer-4 [COMPRESSED] CHARACTERS  
IS VARYING IS SIZE [FROM integer-3 ] [TO integer-4]  
[COMPRESSED] CHARACTERS [DEPENDING ON data-name-1] } ]  
[ LABEL { RECORD IS { STANDARD  
RECORDS ARE } { OMITTED } ]  
[ DATA { RECORD IS  
RECORDS ARE } (data-name-2)... ]  
[CODE-SET IS alphabet-name]  
[ LINAGE IS { data-name-3 } LINES  
integer-5 ]  
[ WITH FOOTING AT { data-name-4 }  
integer-6 ]  
[ LINES AT TOP { data-name-5 }  
integer-7 ]  
[ LINES AT BOTTOM { data-name-6 }  
integer-8 ]



VALUE OF	[	<u>FILENAME</u> IS	{	data-name-7	}	]
				literal-1		
	[	<u>LIBRARY</u> IS	{	data-name-8	}	]
				literal-2		
	[	<u>VOLUME</u> IS	{	data-name-9	}	]
				literal-3		
	[	<u>SPACE</u> IS	{	data-name-10	}	]
				integer-9		
	[	<u>POSITION</u> IS	{	data-name-11	}	]
				integer-10		
	[	<u>INDEX AREA</u> IS	{	data-name-12	}	]
				integer-11		
	[	<u>DATA AREA</u> IS	{	data-name-13	}	]
				integer-12		
	[	<u>PRINT-CLASS</u> IS	{	data-name-14	}	]
				literal-8		
	[	<u>PRINT-FORM</u> IS	{	data-name-15	}	]
				literal-9		
	[	<u>FILE-CLASS</u> IS	{	data-name-6	}	]
				literal-10		
	[	<u>EXTENT SIZE</u> IS	{	data-name-7	}	]
				literal-11		
	[	<u>RECOVERY-BLOCKS</u> IS	{	data-name-8	}	]
				literal-11		
	[	<u>RECOVERY-STATUS</u> IS	data-name-9]			
	[	<u>DATABASE-NAME</u> IS	data-name-20]			

### ***Syntax Rules***

1. The level indicator, FD, identifies the beginning of a file description. It must precede file-name-1, and begin in area A.
2. The clauses following file-name-1 can appear in any order.
3. A file description entry must have at least one record description following it. Each record description specifies a record type for the file.

## **File Description Clauses**

The file description clauses are presented in alphabetical order in the following sections.

### **BLOCK CONTAINS Clause**

The BLOCK CONTAINS clause specifies the size of a physical record or block.

#### ***General Rules***

1. The BLOCK CONTAINS clause is not required. The default assignment is equal to the record size.
2. If a BUFFER SIZE or a RESERVE clause is specified in the corresponding File Control entry, the BLOCK CONTAINS clause specifies the number of records or characters allocated for the buffer; otherwise, this clause is treated as a comment for all but TAPE files.
3. If integer-1 is not specified, integer-2 represents the exact size of the physical record. If both integer-1 and integer-2 are specified, they refer to the minimum and maximum size of the physical record, respectively. Using integer-1 does not imply the existence of a variable-length record, however, this is done with the RECORD CONTAINS clause.
4. The RECORDS phrase specifies the physical record in terms of the logical records. Padding and other factors may affect the actual physical record size used.
5. The CHARACTERS phrase specifies the physical record size in terms of the number of character positions required to store the physical record.

### **CODE-SET Clause**

The CODE-SET clause specifies the character code convention used to represent data on external media. Wang COBOL 85 treats this entry as a comment.

## DATA RECORDS Clause

The DATA RECORDS clause documents the names of the data records of a file. This clause is scheduled to be deleted from the next revision of the Standard.

### *Syntax Rule*

Data-name-2 is the name of a record type and must be defined with a level 01 record description.

### *General Rule*

Specifying more than one data-name means that the file contains more than one type of data record. These records can be of different sizes and different formats. The order in which the names are written in this clause is not significant.

## EXTERNAL Clause

The EXTERNAL clause specifies that a file is external. An external file is available to any program in the run unit that describes it.

### *General Rules*

1. In a run unit, if two or more programs describe the same external file, the record-name of the first record description entry for each of the files must be identical and the records must define the same number of standard data format characters. Subsequent record descriptions, which implicitly redefine the first record, need not be identical. However, the number of character positions in these records must not be greater than the number of character positions of the first record.
2. Even though a file is external, its name is not necessarily global.
3. If an external file contains a LINAGE clause, all redefinitions must have an identical clause, and all data names associated with the external file using the LINAGE clause must also be external.

## GLOBAL Clause

The GLOBAL clause declares a file-name as global. A global file-name is available to the program and to any program contained in that program.

### *Syntax Rules*

1. If a file is specified in a SAME RECORD AREA clause, neither it nor any of its record types can be global.
2. A record that is associated with a global file must have a data-name; it cannot be FILLER.

### *General Rules*

1. A file-name with a GLOBAL clause is a global file-name. Therefore, all subordinate data-names, as well as their associated condition-names, are global names.
2. If program X declares a global name, any program that program X contains, directly or indirectly, can reference that global name without having to declare it.

## LABEL RECORDS Clause

The LABEL RECORDS clause specifies the presence or absence of labels. This is an obsolete element and is scheduled to be deleted from the next revision to the Standard.

### *General Rules*

1. This clause is treated as a comment for all but TAPE files. If this clause is omitted, the STANDARD phrase is assumed.
2. The OMITTED phrase indicates that the file does not have labels.
3. The STANDARD phrase indicates the file has labels. Tape labels conform to ANSI specifications.
4. If a file is external and there is more than one LABEL RECORDS clause associated with it, they must either all specify STANDARD or all specify OMITTED.

## RECORD Clause

The RECORD clause specifies (1) the number of character positions in a fixedlength record or (2) the minimum and maximum number of character positions in a variable-length record.

The size of a data record is specified in terms of the number of character positions required to store the logical record, regardless of the type of characters used to represent the items within that logical record. The size of a record is the sum of the characters in all fixed-length elementary items plus the maximum number of characters in any variable-length item subordinate to the record

In the following rules, the term character positions is used instead of "characters" since it is possible that a character position, an implicit filler for example, does not contain a

character. In Wang COBOL implementations (and most others), a character position is equivalent to a byte.

### ***Syntax Rules***

1. COMPRESSED is not allowed for relative files.
2. Integer-4 must be greater than or equal to integer-3.
3. Data-name-1 must be an elementary unsigned integer defined in the Working-Storage section or the Linkage section.

### ***General Rules***

1. If a RECORD clause is not specified, the size of each record type is defined by its record description entry.
2. The number of character positions indicated by a record description is the sum of the character positions in all elementary data items, excluding data items that have been redefined or renamed, plus any implicit FILLER used for synchronization.
3. The COMPRESSED phrase specifies compressed records. A compressed record is stored as a variable-length record and is treated as such in I-O operations. To create a compressed file, this phrase is required; it is optional when the file already exists.

When a record is compressed, any string containing from 3 to 128 identical consecutive characters is compressed into two bytes; the first byte contains the number of occurrences of the character, and the second byte contains the character itself. The first byte is called the compression control byte.

If a string of 3 to 128 or more non-repetitive characters occurs, the compression control byte contains the number of non-repetitive characters followed by the characters themselves. It is possible for a VS compressed file to be larger than an uncompressed file.

Compression is transparent to the program and programmer. When accessed, compressed records are expanded to the size specified in the record description entry for the file. Thus, the program data area always contains non-compressed records. When "record size" is used in this guide, it always refers to the non-compressed size.

Compressed sequential files may be written, read, and extended but not modified. The COMPRESSED clause on a sequential output file disables any future REWRITE to that file.

4. If integer-3 is omitted from the RECORD CONTAINS clause, integer-4 indicates the number of character positions in each record of the file. (If the file has more than one record descriptions, the record descriptions must all designate integer-4 character positions.)
5. For variable-length records, integer-3 indicates the fewest character positions in a record and integer-4, the most.

For VARYING SIZE records, if integer-3 is not specified, the minimum number of character positions is the number of character positions indicated by the smallest record description for the file. If Integer-4 is not specified, the maximum number of character

positions is equal to the number of character positions indicated by the largest record description for the file.

6. If the record contains a table, the smallest valid number of table elements is used in determining the minimum number of character positions for the record, and the largest valid number of table elements is used in determining the maximum number of character positions for the record.
7. The number of character positions participating in a RELEASE, REWRITE, or WRITE statement is determined by data-name-1. If data-name-1 is specified, it must contain the number of character positions in the current record whenever a RELEASE, REWRITE, or WRITE statement is executed for the file. If data-name-1 is not specified, all the character positions in the record named by these verbs participate.

If the record contains a variable occurrence data item, the number of positions participating is the sum of the fixed portion of the record and the number of currently valid table elements in the record.

8. If a READ or RETURN statement is successfully executed for the file, the contents of data-name are set to indicate the number of character positions in the record just read.

If the READ or RETURN statement has the INTO phrase, the number of character positions participating in the move is determined by the new value of data name-I. If data-name-1 is not specified, all of the character positions in the record participate in the READ or RETURN statement.

9. The value of data-name-1 is not changed by the execution of a DELETE, RELEASE, REWRITE, or WRITE statement or by the unsuccessful execution of a READ or RETURN statement.
10. If an attempt is made to write a record whose character positions are not within the range of integer-3 to integer-4, the operation is unsuccessful; and, unless it was a RELEASE statement, the I-O status is set to indicate this inconsistency.

## VALUE Clause

The VALUE clause provides an initial value to an item associated with the file connector that can be used by the system at runtime. (Alternatively, this information can come from a procedure or by responding to a prompt at runtime.) This clause also specifies the packing density of data blocks for indexed files.

### *Syntax Rule*

Data-name-7 through data-name-20 must be defined in the Working-Storage section. They can be qualified, but they cannot be subscripted, indexed, or have a usage of INDEX.

The following table shows the valid ranges of data-name-7 through data-name-20, and literal-1 through literal-6:

**Value Clause Parameters**

Parameter	Type	Range
data-name-7 literal-1	alphanumeric nonnumeric	1 to 30 <sup>a</sup>
data-name-8 literal-2	alphanumeric nonnumeric	1 to 30 <sup>a</sup>
data-name-9 literal-3	alphanumeric nonnumeric	1 to 30 <sup>b</sup>
data-name-10 integer-9	numeric	1 to 16,777,215
data-name-11 integer-10	numeric	1 to 65,535
data-name-12 integer-11	numeric	1 to 100
data-name-13 integer-12	numeric	1 to 100
data-name-14 literal-4	alphanumeric nonnumeric	1
data-name-15 integer-13	numeric	0 to 999
data-name-16 literal-5	alphanumeric nonnumeric	1
data-name-17 integer-14	numeric	1 to 65,535
data-name-18 literal-6	alphanumeric nonnumeric	1
data-name-19	alphabetic or alphanumeric	1
data-name-20	alphabetic or alphanumeric	1 to 6
<sup>a</sup> Only the first 8 characters are used by the system.		
<sup>b</sup> Only the first 6 characters are used by the system.		

## General Rules

1. The following table shows the file types, OPEN mode, and meaning for each of the parameters in the VALUE clause.

**Value Clause File Types**

Parameter	File Type (device/org.)	OPEN Mode	Meaning
filename	all	all	Filename
library	all	all	Library name
volume	all	all	Volume name
space	all	output	Estimated number of records
position	tape	all	Ordinal position of the file on tape
index area	indexed	output	Index packing density (%) <sup>a</sup>
data area	indexed	output	Data packing density (%)
print-class	printer	output	Print class
print-form	printer	output	Form number
file-class	all	output	File Security Class
extent-size	indexed	output I-O	Number of Blocks to allocate <sup>a</sup>
recovery-blocks	indexed	all	Recovery blocks allocated for transaction
recovery-status	indexed	all	Type of transaction recovery possible
database-name	indexed	all	Database file is attached to <sup>a</sup>

<sup>a</sup> Treated as a comment by the COBOL ReSource.

2. If the file is an output file, the values supplied in this clause are used to complete the file labels.
3. If the file is an input file, the values for data-name-7, data-name-8, and data-name-9 (or the corresponding literals) are used to locate the file at runtime.
4. The INDEX AREA and DATA AREA phrases can be used only with indexed files. If it is omitted, the packing density defaults to 100 percent. The INDEX AREA clause is treated as a comment by the COBOL ReSource.
5. The PRINT-CLASS phrase can be used only with printer files. Only the first character of data-name-4 or literal-4 is used. The default value is A.
6. The PRINT-FORM phrase can be used only with printer files. The value of data-name-15 or integer-13 must be from 0 to 254. The default value is 0.
7. In the FILE-CLASS phrase, only the first character of data-name-16 or literal-5 is used to determine the file class. The default file class is the number sign (#).
8. The EXTENT SIZE phrase specifies the size in blocks for secondary indexed file extents. This physical allocation guidance applies to Wang VS systems only.



9. The possible VS COBOL values of the RECOVERY-BLOCKS phrase literal or data item and their meanings are as follows:

Value	Meaning
N	No recovery blocks
A	Recovery blocks are allocated, but the file is unattached.
U	Recovery blocks are allocated and the file is part of a database.

10. The possible COBOL ReSource values of the RECOVERY-BLOCKS phrase literal or data item and their meanings are as follows:

Value	Meaning
N	File is not recoverable after a soft crash
A	File is recoverable after a soft crash.
U	File is recoverable after a soft crash.

11. When creating a recoverable file, the value A must be in the RECOVERYBLOCKS phrase and the file must be opened in OUTPUT mode. The data management system ignores any other value for new files. If data-name-18 is specified, opening an existing file in any mode but OUTPUT will extract the file's recoverability attribute.

12. The possible values for the RECOVERY-STATUS phrase data-name-19 that are extracted when the file is opened and their meanings are as follows:

Value	Meaning
N	No transaction recovery
S	Soft-crash transaction recovery
F	Full transaction recovery
<b>Note:</b> Full transaction recovery is not available on the COBOL ReSource.	

## LINAGE Clause

The LINAGE clause defines a logical printed page by specifying the number of lines on the page, the size of the top and bottom margins, and the line at which the footing area begins.

### *Syntax Rules*

1. Data-name-3, data-name-4, data-name-5, and data-name-6 must reference unsigned numeric integer data items and may be qualified.
2. Integer-6 or data-name-4 must be greater than zero and cannot be greater than integer-5.
3. Integer-7 or data-name-5 and integer-8 or data name-6 can be zero.

## ***General Rules***

1. The LINAGE clause specifies the size of a logical printed page in terms of the number of lines on that page. The size of a logical page and the size of a physical page are not necessarily the same, although they usually are. The logical page size is the sum of integer-5 or data-name-3, integer-7 or dataname-5, and integer-8 or data-name-6. Each logical page is contiguous to the next logical page; there is no additional spacing save that of the top and bottom margins.
2. Integer-5 or data-name-3 specifies the number of lines that can be written on a logical page. This value must be greater than zero. The part of the logical page that contains these lines is called the page body.
3. Integer-6 or data-name-4 indicates the line in the page body at which the footing area begins. The footing area is the area of the page body from the line indicated by integer-6 or data-name-4 to the line indicated by integer-5 or data-name-3.
4. Integer-7 or data-name-5 specifies the number of lines in the top margin of the page.
5. Integer-8 or data-name-6 specifies the number of lines in the bottom margin of the page.
6. If integer-5, integer-6, integer-7, or integer-8 is specified when a file is opened in OUTPUT mode, the integer determines the number of lines in each of its associated sections of a logical page. These values **are used for all** logical pages written for the file during the execution of the program.
7. If data-name-3, data-name-4, data-name-5, or data-name-6 is specified when a file is opened in OUTPUT mode and a WRITE statement with the ADVANCING PAGE phrase is executed, or when the page overflow condition becomes true, the current value of this data item determines the number of lines in each of its associated sections for the next logical page.
8. If the LINES AT TOP or the LINES AT BOTTOM phrase is not specified, the corresponding margin does not exist.
9. If the FOOTING phrase is not specified, the end-of-page condition will be true only when the page overflow condition is true.
10. If a LINAGE clause is specified, a special register called LINAGECOUNTER is generated. The value of LINAGE-COUNTER is the line number within the page body at which the device is currently positioned. A separate LINAGE-COUNTER is generated for each file whose file description contains a LINAGE clause.
11. LINAGE-COUNTER can be referenced in Procedure Division statements as a "read only" item; only the input-output control system can change its value. If there is more than one LINAGE-COUNTER in a program, it must be qualified by its filename.
12. When an OPEN statement with the OUTPUT phrase is executed for a file, its LINAGE-COUNTER is set to 1.

13. When a WRITE statement to a file is executed, the value of LINAGECOUNTER is modified as follows:
- If an ADVANCING phrase is not specified in the WRITE statement, LINAGE-COUNTER is incremented by 1.
  - If an ADVANCING phrase is specified in the WRITE statement, LINAGE-COUNTER is incremented by the value specified in that phrase.
  - Whenever the device is positioned to the first line of a new logical page, LINAGE-COUNTER is set to 1.
  - If an ADVANCING PAGE phrase is specified in the WRITE statement, LINAGE-COUNTER is implicitly incremented to exceed the value of integer-5 (or data-name-5) and is then set to 1.
14. If the file is external, and if any file description entry has a LINAGE clause, all file description entries must have one. In addition, all the file descriptions that describe the external file must have the same value for integer-5, integer-6, integer-7, and integer-8 and the same external data items for data-name-3, data-name-4, data-name-5, and data-name-6.

## SORT-MERGE File Description Entry

A SORT-MERGE file description entry contains information about the structure of a sort or merge file and the characteristics of the file's data records.

### Format

```
SD file-name-1
[
  BLOCK CONTAINS [integer-1 TO] integer-2 { RECORDS
  CHARACTERS } ]
[
  RECORD { CONTAINS [integer-3 TO] integer-4 [COMPRESSED] CHARACTERS
  IS VARYING IN SIZE [FROM integer-3 ] [TO integer-4]
  [COMPRESSED] CHARACTERS [DEPENDING ON data-name-1] } ]
[
  DATA { RECORD IS
  RECORDS ARE } {data-name-2}... ]
[
  VALUE OF { [
    VOLUME IS { data-name-3 }
    literal-1 ]
    [
    SPACE IS { data-name-4 }
    literal-2 ] } ]
```

### Syntax Rules

1. The level indicator, SD, identifies the file as a sort-merge file.
2. The clauses following file-name-1 are optional and can be in any order.
3. Data-name-1 through data-name-4 can be qualified.

### General Rule

The rules for the clauses of this entry are identical to those for the same clauses in a normal file description entry.

## Working-Storage Section

The Working-Storage section contains descriptions of records and data items that are not part of files. The Working-Storage section consists of a section header followed by entries that describe both records and noncontiguous data items.

In the Working-Storage Section, successive level 01 entries are not implicit redefinitions of the first.

## Initial Values

Unlike data items in files, an item in working storage can have defined initial values. Initial values are assigned by the VALUE clause. The initial value of an index data item and any data item not having a VALUE clause is undefined. The initial value, when defined, can be reset using the INITIALIZE statement.

## Linkage Section

The Linkage section describes data that a calling program makes available to a called program. This section is included in the Data Division of the called program to describe received data.

The structure of the Linkage section is the same as that described for the WorkingStorage section, that is, a section header followed by noncontiguous data items and/or record description entries. However, the VALUE clause is valid in the Linkage section only in condition-name (level 88) entries.

The mechanism by which correspondence is established between the data items in the Linkage section and those in the calling program is explained in the discussions of the Procedure Division Header and the CALL statement. No such correspondence is established for index-names; however, index-names in the called and calling programs always refer to separate indexes.

If a data item defined in the Linkage section whose address is not well defined by one of the above mechanisms, is accessed, the consequences are not predictable.

Normally, a data item defined in the Linkage section is specified in the USING phrase of the Procedure Division header of the called program. A *based record* is a level 01 data item that, although defined in the Linkage section, does not appear in the Procedure Division header USING phrase. Consequently, it has no storage allocated for it, and therefore no address. A based record is given an address by the SET statement. Since it must be an address, it can be SET only to a pointer data item. Pointer data and based addresses are Wang extensions.

The Linkage section may describe both records and noncontiguous data. Records in the Linkage section that are referenced in the Procedure Division header must not contain a REDEFINES clause. In the Linkage Section, successive level 01 entries are not implicit redefinitions of the first level 01 entry.

## External Objects and Internal Objects

The storage associated with a data item or a file can be external or internal to the program in which the item is declared.

A data item or a file is external if the storage associated with that object is associated with the run unit rather than with any particular program within the run unit. An external object may be referenced by any program in the run unit that describes the object. Since there is only one representation of an external object in a run unit, references to an external object from different programs are always to the same object, even though the programs may have separate descriptions of the object.

An item is internal if the storage associated with that object is associated only with the program that describes the item.

*Note: An item, whether internal or external, can have either a global or a local name.*

A record in the Working-Storage section is external if it has the EXTERNAL clause in its description. A data item subordinate to an external record is also external. If an item is not external, it is internal to the program in which it is described.

A file is external if it has an EXTERNAL clause in its file description entry. If a file is external, all its records and subordinate data items are external. If the file is not external, it is internal. A sort-merge file and its subordinate data items, as well as records and subordinate data items described in the Linkage section, are always internal.

## Local Names and Global Names

A filename or a data name is either local or global. A local name can be referenced only from within the program in which the local name is declared. A global name can be referenced either from within the program in which the global name is declared or from within any other program that is directly or indirectly contained in the declaring program.

A name described by using a GLOBAL clause is a global name. A data-name subordinate to a global name is a global name. A condition-name associated with a global name is a global name.

In general, if a data-name, record-name, or file-name is not declared GLOBAL, the name is local.

## Scope of Names

**Programs that are directly or indirectly** contained within other programs can use identical user-defined words to name objects. Although different programs in a run unit may thus use identical names to reference different objects, there is no ambiguity. If an object is referenced by a name that duplicates one or more names in other programs, the reference is to the object in the program where the reference is made.

Other rules governing references are

1. A paragraph-name or section-name can be referenced only by statements in the program in which it is declared
2. Library-names and text-names can be referenced by any program if they are known to the system.
3. An alphabet-name, class-name, condition-name, mnemonic-name, or symbolic character can be referenced by statements and entries in the program that contains the Configuration section in which it is specified and in any program contained in that program.
4. There are special conventions governing data-names, file-names, indexnames, program-names, and record-names, as well as those condition-names that are not specified in the Configuration section. These are discussed in the appropriate sections of this guide.

# Chapter 6 Procedure Division

## Introduction

The Procedure Division, which is optional, describes the operational part of the COBOL program. It is comprised of a hierarchical structure of sections, paragraphs, sentences, and statements. The Procedure Division may also contain complete COBOL source programs that are nested within the COBOL program of which it is a part.

## Statements and Sentences

A statement is a syntactically correct combination of words and symbols beginning with a COBOL verb. A sentence consists of one or more statements terminated by a period

This distinction can be quite significant when control passes to the next sentence rather than to the next statement, for example with the phrase NEXT SENTENCE.

The four types of statements are conditional, declarative, imperative, and delimited scope.

## Conditional Statements

A condition is a state of a program at execution time for which a truth value can be determined. A conditional statement specifies that the truth value of a condition be determined and that subsequent action is dependent on this truth value. The action may consist of a sequence of imperative statements. The conditional statements are

- EVALUATE, IF, PERFORM, and SEARCH statements
- Arithmetic statements with SIZE ERROR phrases
- MOVE WITH CONVERSION statements with the ON ERROR phrase
- Input-Output statements with exception handling phrases
- CALL statements with exception handling phrases

## Declarative Statements

Declarative USE statements indicate specific actions to be performed for exception handling.

## Imperative Statements

An imperative statement is a statement that specifies an unconditional action to be taken or a conditional statement that is delimited by an explicit Scope Terminator. An imperative statement can also consist of a sequence of imperative statements. The verbs that form imperative statements are

ACCEPT	EXIT	RELEASE
ADD	FREE	REWRITE
ALTER	GO	ROLLBACK

CALL	HOLD	SET
CANCEL	INITIALIZE	SORT
CLOSE	INSPECT	START
COMMIT	MERGE	STOP
COMPUTE	MOVE	STRING
CONTINUE	MULTIPLY	SUTRACT
DELETE	OPEN	UNSTRING
DISPLAY	PERFORM	WRITE
DIVIDE	READ	

*Note: These verbs are imperative only if they do not have any optional conditional phrases.*

## Delimited Scope Statements

A delimited scope statement is a statement that includes an explicit Scope Terminator. Scope Terminators terminate a statement within a group of statements without terminating the whole construct. Scope Terminators take the form END-verb. The explicit Scope Terminators are

END-ADD	END-FREE	END-REWRITE
END-BEGIN	END-HOLD	END-ROLLBACK
END-CALL	END-IF	END-SEARCH
END-COMMIT	END-MOVE	END-START
END-COMPUTE	END-MULTIPLY	END-STRING
END-DELETE	END-PERFORM	END-SUBTRACT
END-DISPLAY	END-READ	END-UNSTRING
END-DIVIDE	END-RECEIVE	END-WRITE
END-EVALUATE	END-RETURN	

If a statement without an explicit Scope Terminator is contained (or nested) in another statement, its scope is implicitly terminated by the period at the end of the containing statement or a phrase of the containing statement that follows it.

A conditional phrase in a set of nested statements is associated with the most recently preceding unterminated verb with which that phrase can be associated

## Organization

The Procedure Division consists of a Procedure Division header, followed by the body of the division. The structure of the body of the Procedure Division varies depending on whether or not it has declaratives.



**Format**

```
PROCEDURE DIVISION [ USING { data-name-1 } ... ] .  
  
[ DECLARATIVES .  
  
  { section-name SECTION.  
  
    use-statement.  
  
    [ paragraph-name .  
  
      [ sentence ] ... ] ... } ...  
  
END DECLARATIVES .]  
  
{ section-name SECTION.  
  
  [ paragraph-name .  
  
    [ sentence ] ... ] ... } ...
```

*General Rules*

1. The USING phrase is necessary if and only if the program is called by a CALL statement that has a USING phrase.
2. The USING phrase identifies the names used by the called program for any data items passed to it by a calling program. The items passed are those in the USING phrase of the CALL statement in the calling program.
3. The correspondence between the names in the USING phrase of the CALL statement and the USING phrase of the Procedure Division header is established on a positional basis and not a name basis. Thus, the first data-name in the USING phrase of the calling program corresponds to the first dataname in the USING phrase of the called program's header, the second name to the second name, and so on.
4. A given data-name can appear only once in the USING phrase.
5. Data-name-1 must be an 0 1 or 77 level entry in the Linkage section. It cannot contain a REDEFINES clause in its description, although it can be the object of a REDEFINES clause.
6. A data item defined in the Linkage section of a program can be referenced from elsewhere in the program if
  - It is an operand of the USING phrase of the Procedure Division header or subordinate to such an operand.
  - It is defined with a REDEFINES clause, the object of *which* is an operand of the USING phrase of the Procedure Division header or is subordinate to such an item.
  - It is a condition-name or an index-name associated with a data item that satisfies any of the previous conditions.
  - The program establishes its address by using a SET statement.

7. Declarative sections must be at the beginning of the Procedure Division. They are preceded by the word DECLARATIVES and end with the words END DECLARATIVES. The words DECLARATIVES and END DECLARATIVES must begin in area A and be followed by a period. No other text can appear on the same line as they.

*Note: The Standard specifies that if any paragraph is in a section, all paragraphs must be in sections. As an extension, this rule is not enforced.*

## Arithmetic

This section explains the elements used to express and evaluate arithmetic operations.

### Arithmetic Operators

There are five binary and two unary arithmetic operators represented by the following characters:

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

Unary Arithmetic Operator	Meaning
+	No effect
-	The effect of multiplying the value by -1

Arithmetic operators must always be preceded and followed by a space. However, the two asterisks in the exponentiation operator must be contiguous.

## Arithmetic Expressions

An arithmetic expression is any of the following descriptions:

- An identifier that indicates a numeric item
- A numeric literal
- The figurative constant ZERO
- Any two of the above combined by a binary arithmetic operator
- Two arithmetic expressions separated by a binary arithmetic operator
- An arithmetic expression enclosed in parentheses
- An arithmetic expression preceded by a unary operator

## Arithmetic Expression Evaluation Rules

The order of evaluation in an arithmetic expression is from first to last as shown:

1. Expressions within parentheses are always evaluated first. Nested parentheses are evaluated from the innermost set to the outermost.
2. Unary plus and minus.
3. Exponentiation.
4. Multiplication and division.
5. Addition and subtraction.
6. If an expression has two or more operators at the same level, the operators are evaluated from left to right as they appear in the expression.
7. If the value of an expression that is to be raised to a power is zero, the exponent must be greater than zero or the SIZE ERROR condition exists.
8. If the result of exponentiation can be both a positive and a negative real number, as when evaluating  $(4 ** .5)$ , the result is the positive number. If the result does not yield a real number, as when evaluating  $(-4 ** .5)$ , the SIZE ERROR condition exists.

## Composite of Operands

Arithmetic expressions permit the combining of arithmetic operations without the restrictions on composite of operands and/or receiving data items. Refer to Appendix G for details on the precision of intermediate results in arithmetic expressions.

*Note: The arithmetic statements are ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT. For all arithmetic statements except COMPUTE, neither an operand nor the composite of operands can exceed 18 decimal digits.*

The composite of operands is a hypothetical data item resulting from the superimposition of operands specified in a statement aligned by their decimal points.

The operands of arithmetic statements need not have the same usage; any necessary conversion and decimal point alignment is performed automatically.

## Truncation and Rounding

During an arithmetic operation, after the result and the receiving item are aligned by decimal point, the number of places in the fractional part of the result can be greater than the number of places in the receiving item. When this occurs, the result is truncated to make it fit.

When the **ROUNDED** phrase is specified in an arithmetic statement before truncation occurs, the most significant digit of the part to be truncated is examined. If it is a five or more, the absolute value of the result is increased by one in the loworder position. Then, the result is truncated.

If any low-order positions of a receiving item are represented by a **P** in the item's **PICTURE** clause, the rounding or truncation occurs relative to the rightmost digit position.

## Multiple Receiving Items in Arithmetic Statements

It is possible for the **ADD**, **COMPUTE**, **DIVIDE**, **MULTIPLY**, and **SUBTRACT** statements to have multiple receiving items. Such statements first access all the data items that go into producing the result, perform any necessary arithmetic on these data items, and store the result in a temporary location.

Then they transfer or combine the value in this temporary location with each individual receiving item in the same left-to-right sequence as they are specified in the statement.

For example, the statement

```
ADD A B C TO C D(C) E
COLUMN NUMBER IS integer-1
```

is equivalent to

```
ADD A B C GIVING TEMP
ADD TEMP TO C
ADD TEMP TO D(C)
ADD TEMP TO E
```

and the statement

```
MULTIPLY A(I) BY I, A(I)
```

is equivalent to

```
MOVE A(I) TO TEMP
MULTIPLY TEMP BY I
MULTIPLY TEMP BY A(I).
```

## Conditions

A conditional expression specifies a condition to be tested. Subsequent program action depends upon whether the condition is true or false.

## Relation Conditions

A relation condition compares two operands, each of which can be a data item, a literal, an arithmetic expression, or an index-name. A relation condition is true if the specified relation exists between the operands.

### Format

$$\left\{ \begin{array}{l} \text{data-name-1} \\ \text{literal-1} \\ \text{expression-1} \\ \text{index-name-1} \end{array} \right\} [\text{IS}] \left\{ \begin{array}{l} [\text{NOT}] \text{ GREATER THAN} \\ [\text{NOT}] \text{ LESS THAN} \\ [\text{NOT}] \text{ EQUAL TO} \\ \text{GREATER THAN OR EQUAL TO} \\ \text{LESS THAN OR EQUAL TO} \end{array} \right\} \left\{ \begin{array}{l} \text{data-name-2} \\ \text{literal-2} \\ \text{expression-2} \\ \text{index-name-2} \end{array} \right\}$$

**Note:** The symbols  $>$ ,  $=$ , and  $<$  are equivalent to *GREATER THAN*, *EQUAL TO*, and *LESS THAN*. There is no difference in the operation if they are used, but  $>=$  and  $<=$  must be used without the word *OR*.

### General Rules

1. The operand on the left is the subject of the condition; the one on the right is the object of the condition.
2. A relation condition must contain at least one reference to a variable.
3. The relational operators (*GREATER THAN*, *LESS THAN*, etc.) specify the type of comparison to be made in a relation condition. Each word (or symbol) in the operator must be preceded and followed by a space.
4. The word *NOT* is considered to be part of the relational operator that follows it.
5. The operator *NOT GREATER THAN* is equivalent to *LESS THAN OR EQUAL TO*; and *NOT LESS THAN* is equivalent to *GREATER THAN OR EQUAL TO*.

### Numeric Comparisons

Numeric operands are compared by their algebraic values; the number of digits or the usage of the particular operand is not significant. Unsigned numeric operands are considered positive for purposes of comparison, and zero is considered a unique value regardless of the sign.

### Nonnumeric Comparisons

If one (or both) operands are nonnumeric or if either of the operands is a group item, the comparison is made with respect to the program collating sequence. The size of an operand is the total number of characters in the operand. If the operands are of equal size, characters in corresponding positions are compared, starting from the left, and continuing until either a pair of unequal characters is encountered or the rightmost characters are reached.

If all pairs of corresponding characters are equal, the items are equal. If a pair of unequal characters is found, the characters' relative positions in the collating sequence are compared. The operand with the character that is higher in the collating sequence is the greater operand.

If the operands are of unequal size, the shorter operand is treated as if it were extended on the right by sufficient spaces to make the sizes equal. Comparison then proceeds as for operands of equal size.

In comparing a numeric and nonnumeric operand, The numeric operand must be an integer with a usage of DISPLAY. If the nonnumeric operand is an elementary data item or a nonnumeric literal, the numeric operand is treated as though it were first moved to an elementary alphanumeric data item of the same size, and then this alphanumeric data item is compared to the nonnumeric operand. If the nonnumeric operand is a group item, the numeric operand is treated as though it were first moved to a group item of the same size, and then this group data item is compared to the nonnumeric operand.

### **Comparison of Index-Names and/or Index Data Items**

Comparisons involving index-names and/or index data items can be made only between

Two index-names - The corresponding occurrence numbers are compared

An index-name and a nonindex data item or literal - The occurrence number of the index is compared with the data item or literal.

Two index data items or an index data item and an index-name or another index data item - Their actual contents are compared without regard for the occurrence numbers represented.

## **Class Conditions**

A class condition is a test of the class of the contents of an operand. An operand can be either numeric, alphabetic, alphabetic-lower(case), alphabetic-upper(case), or any class specified in the CLASS clause in the SPECIAL-NAMES paragraph.

Format

```
data-name-1 IS [NOT]
                ALPHABETIC
                ALPHABETIC-UPPER
                ALPHABETIC-LOWER
                NUMERIC
                class-name-1
```

The class of the contents of an operand is determined as follows:

- The contents are numeric if they consist entirely of the characters 0, 1, 2, 3,4, 5, 6, 7, 8, and 9, with or without an operational sign.
- The contents are alphabetic if they consist entirely of the uppercase letters A to Z and space, or the lowercase letters a to z and space, or any combination of the uppercase and lowercase letters and spaces. (This differs from VS COBOL 74 where only uppercase characters and the space character are considered alphabetic.)
- The contents are alphabetic-lower if they consist entirely of the lowercase letters a to z and space.
- The contents are alphabetic-upper if they consist entirely of the uppercase letters A to Z and space.
- The contents belong to class-name-1 if they consist entirely of the characters specified in the definition of class-name-1 in the SPECIAL-NAMES paragraph.

### ***Syntax Rule***

The usage of the operand being tested must be DISPLAY.

### ***General Rules***

1. When used, NOT and the next keyword specify one class condition that defines the class test to be executed for truth value. In other words, NOT NUMERIC is a truth test for determining that the contents of an operand are not numeric.
2. The NUMERIC test is true if the contents of the item are numeric and an operational sign is present if and only if the item's data description indicates such a sign. This test cannot be used with an alphabetic item or with a group item composed of elementary items that have operational signs. In Appendix B, the description of the Compiler option SEPSGN has more information on signs.
3. The ALPHABETIC test is true if the contents of the item consist entirely of alphabetic characters and spaces. This test cannot be used with a numeric item.
4. The ALPHABETIC-LOWER test is true if the contents of the item consist entirely of lowercase alphabetic characters and spaces. This test cannot be used with a numeric item.
5. The ALPHABETIC-UPPER test is true if the contents of the item consist entirely of uppercase alphabetic characters and spaces. This test cannot be used with a numeric item.
6. The class-name-1 test is true when the contents of the item consist entirely of the characters defined as belonging to class-name-1. This test cannot be used with a numeric item.

## **Condition-Name Tests**

A condition-name tests a conditional variable to determine whether or not its value is equal to one of a set of values.

### **Format**

condition-name-1

### ***General Rules***

1. If condition-name-1 has a single value, the test is true if the current value of the conditional variable equals the value of the condition-name.
2. If condition-name-1 has a range of values, the test is true if the current value of the conditional variable value is within this range, including the end values.
3. The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.
4. Condition-name-1 may name a switch-status condition to determine the on or off status of a switch. The switch and its associated ON or OFF values are specified in the SPECIAL-NAMES paragraph. The test is true if the switch is set to the position corresponding to condition-name-1.

## Sign Condition

The sign condition determines whether or not the algebraic value of an arithmetic expression is less than, greater than, or equal to zero.

### *Format*

```
arithmetic-expression-1 IS [NOT] POSITIVE
                                NEGATIVE
                                ZERO
```

### *General Rules*

1. The word NOT in combination with the next keyword specifies one sign condition that defines the algebraic test. For example, NOT ZERO is a truth test for a nonzero (that is, positive or negative) value.
2. The test for POSITIVE is true if the expression is greater than zero.
3. The test for NEGATIVE is true if the expression is less than zero.
4. The test for ZERO is true if the expression is equal to zero.
5. Arithmetic-expression-1 must contain at least one reference to a variable.

## Modified Data Tag Conditions

Whenever a user modifies a field of a workstation screen, bit 1 of that field's FAC is set to 1. This bit is called the modified data tag (MDT). The modified data tag condition tests this bit to determine if the field has been modified.

### *Format*

```
FAC OF display-data-item ALTERED
```

### **General Rule**

The FAC OF test is true if bit 1 of the data item's FAC is 1.

## Mask Tests

A user-figurative-constant can be used to test a bit or combination of bits in any one-byte item.

### *Format*

```
user-figurative-constant      IN identifier      ON
                                OF FAC OF display-item  IS [NOT] OFF
```

### *General Rules*

1. The ON condition is true if the bits set to 1 in the FAC or data-name correspond to the bits set to 1 in the user-figurative-constant.



2. The OFF condition is true if the bits set to 0 in the FAC or data-name correspond to the bits set to 1 in the user-figurative-constant.

For example, with the entries

```
FIGURATIVE-CONSTANTS ONE IS "01".
```

```
77 OPR-1 PICTURE 9 VALUE 1.
```

the condition

```
IF ONE IN OPR-1 IS ON
```

is true.

3. User-figurative-constant and data-name must be 1-byte items.

## Simple and Complex Conditions

Simple conditions can be combined by the logical operators AND, OR, and NOT. A complex condition is (1) two simple or complex conditions combined by the binary operators AND and OR, or (2) a simple or complex condition that is negated by the unary operator NOT. The logical operators must always be *preceded and followed by a space*. A complex condition can be put in *parentheses*. The truth value of a complex condition, whether parenthesized or not, is determined by the truth values of its components with respect to the logical operator used.

There are two types of complex conditions: negated conditions and combined conditions

### Negated Condition

A negated condition is one that has a NOT operator.

#### Format

```
NOT condition-1
```

#### General Rule

A negated condition is true if the original condition is false and is false if the original condition is true.

### Combined Condition

A combined condition is two conditions connected by AND or OR.

#### Format

```
condition-1  AND  condition-2 ...  
              OR
```

#### General Rules

1. If two conditions are combined by AND, the combined condition is true if both of the constituent conditions are true and is false if one or both of them is false.
2. If two conditions are combined by OR, the combined condition is true if either or both of the constituent conditions are true and false if both of them are false.

## Abbreviated Combined Relation Conditions

Simple and negated simple relation conditions can be written in an abbreviated form, if

- The conditions are combined with logical connectives in a consecutive sequence.
- A succeeding relation condition contains a subject or a subject and a relational operator that are common with the preceding relation condition.
- No parentheses are used within the sequence.

The abbreviation can omit the subject of the relation condition or both the subject and relational operator of the relation condition.

*Note: The use of abbreviated combined relation conditions tends to obscure the program logic. Because they offer no increase in execution speed, they are best avoided.*

### Format

relation-condition    **AND** [NOT] [relational-operator] object ...  
                                  **OR**

### General Rules

1. The effect of using these abbreviations is as if the last preceding stated subject were inserted in place of the omitted subject, and the last stated relational operator were inserted in place of the omitted relational operator.
2. The implied insertion terminates once a complete, simple condition occurs within the complex condition.
3. The presence of the word NOT in an abbreviated combined relation condition is interpreted as follows:
  - If the word immediately following NOT is GREATER, >, LESS, <, EQUAL, or =, the NOT participates as part of the relational operator.
  - If the above is not true, the NOT is interpreted as a logical operator and, therefore, the implied insertion of subject or relational operator results in a negated relation condition.

## Precedence of Logical Operators

In the absence of parentheses, the hierarchy of logical operators is, from highest to lowest: NOT, AND, and OR. For example, the expression

condition-1 *OR NOT* condition-2 **AND** condition-3

is equivalent to

condition-1 **OR** (( **NOT** condition-2) **AND** condition-3).

Parentheses can be used in a complex condition to override the default hierarchy. In the preceding example, if parentheses are used, the expression can take on a different meaning:

(condition-1 *OR* (*NOT* condition-2)) **AND** condition-3.

## Order of Condition Evaluation

The order of evaluation of conditions within complex conditions is based on explicit and implicit hierarchies. The implicit hierarchy is that mentioned previously: NOT, AND, and OR. An explicit hierarchy is denoted by parentheses.

Therefore, a complex condition can be considered a nested structure of hierarchical levels, with the entire structure itself being the most inclusive level.

Evaluation of a complex condition begins at the left and proceeds according to the following rules, applied recursively where necessary:

1. The constituent connected conditions within a given level evaluation is from left to right.
2. Evaluation of a given level terminates as soon as a truth value for it has been determined, regardless of whether all the constituent conditions within that level have been evaluated.
3. Arithmetic expressions contained within conditions are evaluated when the condition is evaluated.

## Exception Handling Phrases

An exception condition is a state of a program at execution time that, while it may be expected, falls outside the norm. An exception handling phrase specifies a condition to be identified and subsequent action that is dependent on its occurrence. The action may consist of a sequence of imperative statements. If an exception condition occurs, the causing statement is unsuccessful. The exception handling phrases and the statements that support them are shown in the following table:

### Exception Handling Phrases

<b><u>Exception Handling Phrases</u></b>	<b><u>Statements Supporting Them</u></b>
[NOT] INVALID KEY	DELETE, READ, REWRITE, START, WRITE
[NOT] ON SIZE ERROR	Arithmetic Statements
ON ERROR	MOVE WITH CONVERSION, ROLLBACK
PFKEY	DISPLAY AND READ
NOMOD	DISPLAY AND READ
TIMEOUT	HOLD, READ, WRITE
[NOT] AT END	READ, RETURN
[NOT] ON EXCEPTION	CALL
ON OVERFLOW	CALL

When an exception handling phrase is specified and the EXCEPTION condition occurs, or when a NOT Exception phrase is specified, and the EXCEPTION condition does not occur, control is transferred to the relevant imperative-statement. Execution then continues with each statement of the imperative-statement. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of the imperative-statement, control is transferred to the end of the statement for which the exception is detected.

### Arithmetic SIZE ERRORS

Two types of errors are handled by Arithmetic SIZE ERROR phrases. When the absolute value of the result exceeds the largest value that can be contained in a receiving item, the size error condition exists. If the size error occurs during an ADD CORRESPONDING or

SUBTRACT CORRESPONDING statement, imperative-statement-1 is executed only after all of the individual additions or subtractions have been completed.

The SIZE ERROR condition also exists when an attempt is made to divide by zero or when the rules for exponentiation are violated. These two situations also cause the arithmetic operation to terminate, which leaves the values of all receiving items unchanged and, in the absence of a SIZE ERROR phrase, leads to program termination.

If an Arithmetic SIZE ERROR phrase is not specified and a size error occurs, the values of the affected items are incorrect. Where the compiler detects the possibility for this error, it issues a warning on the possible loss of precision.

## Declaratives Execution

If an EXCEPTION condition occurs and an exception handling phrase is not specified, and if there is an applicable declarative USE procedure, it is executed, and upon its completion control returns to the next executable statement following the statement that caused the exception.

## Program Termination

If an EXCEPTION condition occurs that is not handled by an exception handling phrase or a declarative procedure, program execution is terminated.

## Other Considerations

The other considerations that should be taken into account are described in the following sections.

### Overlapping Operands

If the sending and receiving items in a statement share any part of their storage areas but are not defined by the same data description entry, the result of the execution of such a statement is not predictable. In addition, even if sending and receiving items are defined by the same data description entry, the results are not predictable for the STRING, UNSTRING, and INSPECT statements.

### Incompatible Data

With the exception of the class test, if a data item is referenced and its contents are not compatible with its class, the result is undefined

### INVALID KEY Condition

The INVALID KEY condition can occur as a result of the execution of a START, READ, WRITE, REWRITE or DELETE statement.

When the INVALID KEY condition is recognized, these actions are taken in the following order:

1. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an INVALID KEY condition.

2. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any USE procedure specified for this file is not executed
3. If the INVALID KEY phrase is not specified, a USE AFTER EXCEPTION procedure must be associated with the file connector and executed, and control is transferred according to the rules of the USE statement. The NOT INVALID KEY phrase is ignored, if specified

#### **FILE ATTRIBUTE CONFLICT Condition**

The FILE ATTRIBUTE CONFLICT condition can occur as a result of the execution of a OPEN, WRITE, or REWRITE statement. When the FILE ATTRIBUTE CONFLICT condition is recognized, these actions are taken in the following order:

1. A value is placed into the I-O STATUS associated with the file-name to indicate the FILE ATTRIBUTE CONFLICT condition.
2. If a USE AFTER EXCEPTION procedure is associated with the file-name, it is executed.

# Chapter 7 Procedure Division Statements

## Introduction

This chapter explains *the* various COBOL verbs, which are presented in alphabetical order. For information as to how they interact with one another, refer to the VS *Programmer's Guide to COBOL* or the *COBOL ReSource Programmer's Guide*, as appropriate.

## ACCEPT Statement

The ACCEPT statement takes data from a device and puts it into a data item.

### Format

```
mnemonic-name-1
DATE
ACCEPT data-name-1 FROM DAY
TIME
DAY-OF-WEEK
```

### Syntax Rule

Mnemonic-name-1 is specified in the SPECIAL-NAMES paragraph.

### General Rules

1. An ACCEPT statement transfers data either by extracting it from the system or by prompting the operator for the data.
2. ACCEPT statements without a FROM clause or with a FROM mnemonicname- I clause are equivalent.
3. If data-name-1 is numeric, the length of the data entry field displayed on the screen is equal to the number of 9s in the picture string of the item. This field is increased by one position if the picture contains an S and another position if the picture contains a V. If a data item with usage BINARY and no PICTURE clause is specified, a default signed picture is provided by the compiler. The data from the screen is moved into data-name-1 according to the rules for a MOVE WITH CONVERSION statement.
4. If data-name-1 is not numeric, the length of the data entry field is equivalent to the number of characters represented by the item's picture string. The data from the screen is moved into data-name-1 as if both the sending and the receiving fields were described as alphanumeric. Therefore, for edited items, the data must be entered in the edited form. The JUSTIFIED clause does not apply to the operation of the ACCEPT statement.
5. In a prompt (that can be satisfied by a Procedure Language entry), ACCEPT is used as the parameter reference name, and the data name (truncated to eight characters) is used as the receiving field name.
6. Data-name-1 may not have a usage of DISPLAY-WS.
7. Up to 16 separate data items can be accepted in one statement.
8. DATE is extracted from the system as an unsigned numeric integer, and it contains six digits. It consists of the year of the century, the month of the year, and the day of the month (from left to right). Each of these items contains two digits. For example, July 1, 1992 is represented as 920701.
9. DAY is extracted from the system as an unsigned numeric integer, and it contains five digits. It consists of the year of the century and the day of the year (from left to right). The first item contains two digits; the second, three digits. For example, July 1, 1992 is represented as 92183.

10. TIME is extracted from the system as an unsigned numeric integer, and it contains eight digits. It consists of the number of hours, minutes, seconds, and hundredths of a second after midnight (from left to right). Each item contains two digits. TIME is based on a 24-hour clock, so the minimum value of TIME is 00000000; the maximum value, 23595999. For example, 2:41 p.m. is expressed as 14410000.
11. DAY-OF-WEEK is extracted from the system as an unsigned numeric integer, and it contains one digit that represents the day of the week, with 1 being Monday, 2 Tuesday, and so on.



## ADD Statement

The ADD statement forms the sum of two or more operands and stores the result. Refer to Chapter 6 for details about program behavior in the presence of arithmetic SIZE ERROR clauses.

### Format 1

```
      data-name-1
ADD    ..TO {data-name-2 [ROUNDED] }...
      literal-1

[ ON SIZE ERROR imperative-statement-1 ]

[ NOT ON SIZE ERROR imperative-statement-2 ]

[ END-ADD ]
```

### Syntax Rules

1. All operands must be numeric.
2. The composite of operands cannot contain more than 18 digits. Refer to Chapter 6 for additional information.

### General Rule

The values of the operands preceding the word TO are added together, and their sum is stored in a temporary data item. This temporary data item is then added to the current value of data-name-2, and the result is stored in data-name-2. This process of adding the temporary data item is repeated for each data item following TO, the items being treated in the same left-to-right order as they are written in the statement.

### Format 2

```
      data-name-1      data-name-2
ADD    TO
      literal-1      .. literal-2

GIVING {data-name-3 [ ROUNDED ] } ...

[ ON SIZE ERROR imperative-statement-1 ]
[ NOT ON SIZE ERROR imperative-statement-2 ]

[ END-ADD ]
```

### Syntax Rules

1. All operands preceding the word GIVING must be numeric. Operands following GIVING must be numeric or numeric edited.

2. The composite of operands preceding GIVING cannot contain more than 18 digits. Refer to Chapter 6 for additional information on composite of operand.

#### *General Rule*

The values of the operands preceding the word GIVING are added together. This sum is then stored as the new contents of data-name-3. This process of storing the sum is repeated for each data item following GIVING, the items being treated in the same left-to-right order as they are written in the statement.

#### *Format 3*

```
ADD CORRESPONDING data-name-1 TO data-name-2 [ ROUNDED ]  
[ ON SIZE ERROR imperative-statement-1 ]  
[ NOT ON SIZE ERROR imperative-statement-2 ]  
[ END-ADD ]
```

#### *Syntax Rules*

1. Both data-names must be group items.
2. The composite of operands, which is determined separately for each pair of items, cannot contain more than 18 digits. Refer to Chapter 6 for additional information.

#### *General Rule*

Data items in data-name-1 that have a corresponding data item in data-name-2 are added to and stored in that corresponding data item. Refer to Chapter 2 for additional information on corresponding data items.

## ALTER Statement

The ALTER statement modifies a predetermined sequence of operations. This statement has the potential for causing great harm in the form of obscure program errors, and its use is not recommended. In addition, it is an obsolete element and is slated for deletion from the next revision of the Standard.

### *Format*

ALTER {procedure-name-1 TO [PROCEED TO] procedure-name-2}...

### *Syntax Rules*

1. Procedure-name-1 must be a paragraph containing one GO TO statement without the DEPENDING phrase.
2. Procedure-name-2 must be a paragraph or a section name in the Procedure Division.

### *General Rule*

The ALTER statement modifies the GO TO statement in paragraph-1 so that subsequent executions of it transfer control to procedure-name-2. This alteration remains in effect until another ALTER statement is executed for paragraph-1.

## BEGIN Statement

The BEGIN statement indicates the start of a transaction or subtransaction. For information about the creation of transaction-capable files, refer to the VALUE clause discussion of RECOVERY-BLOCKS, RECOVERY-STATUS, and DATABASE-NAME implementor names in Chapter 5.

### Format

```
BEGIN          TRANSACTION  
              SUBTRANSACTION  
  
              [ ON ERROR imperative-statement-1 ]  
              [ NOT ON ERROR imperative-statement-2 ]  
  
[ END-BEGIN ]
```

### Syntax Rule

TRANSACTION and SUBTRANSACTION cause the same actions to be performed. The different words provide a means of documenting the program logic.

### General Rules

1. The BEGIN statement marks the beginning of a transaction; or, if an uncommitted transaction exists, it marks the beginning of a subtransaction. An uncommitted transaction exists in any of the following conditions:
  - A BEGIN statement has been executed but a corresponding COMMIT, ROLLBACK, or FREE ALL statement has not been executed
  - A transaction-capable file is opened and updated but no COMMIT or FREE ALL statement has been issued since the last update.
2. The BEGIN statement is optional for marking the start of a subtransaction, unless the top-level transaction executed a BEGIN statement prior to performing transaction-capable file updates, in which case it is required
3. If the operation is successful, the special register RETURN-CODE is set to zero. If the operation is unsuccessful, a nonzero value is put into RETURNCODE.
4. The ON ERROR phrases, if specified, are handled according to the exception handling rules described in Chapter 6.

## CALL Statement

The CALL statement invokes another program in the run unit.

### Format 1

```
CALL { identifier-1 } [ USING { [ BY { REFERENCE } ] { identifier-2 } ... } ]  
    { literal-1 }  
    [ ON EXCEPTION imperative-statement-1 ]  
    [ NOT ON EXCEPTION imperative-statement-2 ]  
    [ END-CALL ]
```

### Format 2

```
CALL { identifier-1 } [ USING { [ BY { REFERENCE } ] { identifier-2 } ... } ]  
    { literal-1 }  
    [ ON OVERFLOW imperative-statement-1 ]  
    [ END-CALL ]
```

### Syntax Rules

1. Identifier-1 must be alphanumeric.
2. Literal- I must be nonnumeric.
3. An operand in the USING phrase must be a data item in the File section, Working-Storage section, or Linkage section. A file-name cannot be used as a parameter when calling a COBOL subprogram.

### General Rules

1. Literal-1 or the contents of data-name-1 is the name of the called program. The program in which the CALL statement appears is the calling program.
2. When the called program is a COBOL program, the contents of literal- I or data-name-1 must be the program-name specified in the PROGRAM-ID paragraph of the program. When the called program is a C program, written in another language, the contents must be the name of the called function or procedure.
3. On the Wang VS, the number of characters that are significant in the called program name depend on the Compiler option OBJFORM. Refer to Appendix B. If OBJFORM is 0, only the first seven characters of the called program's name are significant and must be unique.
4. When a filename is used as a parameter on the Wang VS, the address of the User File Block (UFB) is passed to the routine.
5. The OVERFLOW phrase is equivalent to the EXCEPTION phrase. For additional information, refer to Chapter 6.

6. If two programs in a run unit have the same name, one of those programs must be directly or indirectly contained within another program that does not contain the other of the two. When this communal name is specified in a CALL statement, the reference is resolved as follows:
  - If one of the programs is directly contained within the program with the CALL statement, it is the one called.
  - If one of the programs contains, directly or indirectly, the program with the CALL statement, it will not be called.
  - If one of the programs is a COMMON program and is directly contained by a program that contains (directly or indirectly) the program with the CALL statement, the common program will be called. Otherwise, the separately compiled program is called.
7. If a called program has the initial attribute, then it, as well as all the programs it directly or indirectly contains, will be in its initial state every time it is called. If the called program does not have the initial attribute, then it, as well as all the programs it directly or indirectly contains, will be in its initial state
  - The first time it is called in the run unit
  - The first time it is called after the execution of a CANCEL statement either for it or for a program that directly or indirectly contains it
8. When a called program is in its initial state, all data items with a VALUE clause are set to the specified value, and any internal files are closed. If it is not in its initial state, the program's state (and the state of all programs that it directly or indirectly contains) is the same as that which existed when the program was last exited. In particular, the states and positioning of all internal files are the same.
9. Neither calling a program nor exiting a program that has been called alters the status or positioning of any external file.
10. When the called program is a COBOL program, the USING phrase is necessary only if there is a USING phrase in the Procedure Division header of the called program. In this case, the number of operands in each USING phrase must be identical. When the program being called is written in a language other than COBOL, the number of operands in the USING phrase of the CALL statement should agree with the number of operands the called program expects to receive.
11. The correspondence between the data items used by the calling and called programs is positional and not by name.
12. The values of the parameters in the USING phrase are made available to the called program. Both the BY CONTENT and BY REFERENCE phrases are transitive across the parameters that follow them until another BY CONTENT or BY REFERENCE phrase is encountered. If neither phrase is specified prior to the first parameter, the BY REFERENCE phrase is assumed.
13. If the BY REFERENCE phrase is either specified or implied for a parameter, the program operates as if the corresponding data items in both programs occupied the same

storage area. Therefore, if a data item is passed by reference, both data item descriptions must specify the same number of character positions.

14. If the BY CONTENT phrase is specified or implied for a parameter, the called program cannot change the value of the parameter specified in the CALL statement's USING phrase. However, the called program can change the value of the corresponding data item specified in its own procedure division USING phrase. Therefore, the data description of each parameter passed by content must be identical to the data description of the corresponding parameter in the USING phrase of the Procedure Division header. No conversion between operands with different usages or extension or truncation of operands of different sizes is performed
15. Called programs can contain CALL statements. However, a called program must not execute a CALL statement that directly or indirectly calls the calling program.
16. If a CALL statement is executed within the range of a declarative, that CALL statement cannot directly or indirectly reference any called program to which control has been transferred and has not completed execution.
17. If LENGTH OF data-name is specified in the USING phrase, the data-name cannot have an OCCURS DEPENDING clause, and it cannot contain a data item with such a clause. It can, however, be subordinate to a data item with an OCCURS DEPENDING clause. In addition, the value is always passed by content; any BY REFERENCE phrase that might be in force has no effect. This does not, however, alter the implicit REFERENCE/CONTENT switch. For example, with the following phrase, both ABC and XYZ are passed by reference:

USING ABC, LENGTH OF DEF, XYZ

18. If ADDRESS OF data-name is specified in the USING phrase, and the data name is the name of a based record and is specified without subscripting or reference-modification, the following action takes place: When it is passed by reference, any changes to the corresponding pointer item in the called program will change the address of the based record in the calling program. If passed by content, such changes are not reflected to the calling program. For all other forms of "ADDRESS OF data-name" the value is passed by content; any BY REFERENCE phrase that might be in force has no effect.
19. The END-CALL phrase delimits the scope of the CALL statement.
20. The special register RETURN-CODE is updated upon return from a called program.

## CANCEL Statement

The CANCEL statement ensures that a program is in its initial state the next time it is called.

### **Format**

$$\text{CANCEL } \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \dots$$

### **Syntax Rules**

1. Data-name-1 must be alphanumeric.
2. Literal-1 must be nonnumeric.

### **General Rules**

1. A program can be cancelled by a program only if it is callable by that program.
2. A program can be cancelled only if the Compiler option CANCEL is YES.
3. A called program can be cancelled by termination of the run unit; by execution of a CANCEL statement referencing it or a program that contains it; or, if the called program has the initial attribute, by execution of an EXIT PROGRAM statement.
4. A program that has been called and has not executed an EXIT PROGRAM statement must not be cancelled, implicitly or explicitly.

When a program is canceled, any programs contained within it are implicitly canceled. The result is the same as if a CANCEL statement had been executed for each contained program, beginning with the one at the innermost level in the explicitly canceled program.
5. When a program is canceled because it is specified in a CANCEL statement, that program is explicitly canceled. If a program is explicitly cancelled, any . program that is contained within it is implicitly canceled.
6. The program to be canceled is the one whose name is literal- 1 or in data item-1. On the Wang VS, the number of characters that are significant in the name depends on the Compiler option OBJFORM. If OBJFORM is 0, only the first seven characters of the called program's name are significant and, therefore, must be unique. If OBJFORM is 1, the first 30 characters are significant and must be unique.
7. When a program is implicitly or explicitly canceled, an implicit CLOSE statement (with no optional phrases) is executed for each open file internal to the program. Any USE procedures associated with these files are not executed, and external data items are not changed.
8. When an implicit or explicit CANCEL statement is executed for a program that either has not been called or that has been called but has already been canceled nothing happens to the program, and control passes to the statement following the CANCEL.
9. The CANCEL statement has no effect on programs in a shared subroutine library.



## CLOSE Statement

The CLOSE statement terminates the processing of reels and files with optional rewinding and/or locking. Note that the operating system attempts to close any open files when a STOP RUN statement is executed

### **Format**

$$\text{CLOSE} \left\{ \text{file-name-1} \left[ \begin{array}{l} \left\{ \begin{array}{l} \text{REEL} \\ \text{UNIT} \end{array} \right\} [\text{FOR REMOVAL}] \\ \text{WITH} \left\{ \begin{array}{l} \text{NO REWIND} \\ \text{LOCK} \end{array} \right\} \end{array} \right] \right\} \dots$$

### **Syntax Rules**

1. The reel (or unit) phrase can be used *only* with tape files.
2. The NO REWIND phrase can be used only with tape files.
3. It is not necessary for files referenced in a CLOSE statement to have the same organization or access.

### **General Rules**

1. The terms reel and unit are synonymous and interchangeable when used with the CLOSE statement and in these rules. Treatment of disk files is logically equivalent to the treatment of tape files.
2. A CLOSE statement can be executed only for an open file.
3. For the purpose of showing the effect of the CLOSE statement on various storage media, files are divided into the following categories:
  - Nonunit. This is a file whose input or output medium is such that the concepts of rewind and unit have no meaning.
  - Sequential single unit. This is a sequential file that is entirely contained on one unit. Treatment of a file contained in a multiple file tape is logically equivalent to the treatment of a file wholly contained on one reel.
  - Sequential multiunit. This is a sequential file that is contained on more than one unit.
4. The results of executing each type of CLOSE for each category of file are summarized in the table shown in the section titled "Explanation of CLOSE Statement Formats" later in this chapter.
5. Execution of the CLOSE statement updates the FILE STATUS data item.
6. When an optional input file is not present, no end-of-file or unit processing is performed for the file. Both the file position indicator and the current volume pointer remain unchanged.
7. Following the successful execution of a CLOSE statement without the UNIT phrase, the file is no longer associated with a file connector. In addition, the file's record area is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undetermined.

8. If more than one file-name is specified in a CLOSE statement, the files are closed in the order in which their names appear in the statement, as if each had had a separate CLOSE statement specified for it.
9. When a CLOSE statement is executed for a file, no statements except SORT and MERGE can be executed for that file, either explicitly or implicitly, unless the file is again opened.
10. Indexed and relative files are classified as belonging to the category of non-sequential single/multiunit. The results of executing each type of CLOSE for this category of file are summarized in the table shown in the section titled "Explanation of CLOSE Statements for File Categories" later in this chapter.
11. Execution of a CLOSE statement for a transaction-capable file with uncommitted transactions will commit those transactions prior to closing the file. Refer to the section titled "COMMIT Statement" for details.

## Explanation of CLOSE Statement Formats

The following table shows the results of executing each type of a CLOSE statement format for each file category:

**File Category**

CLOSE Statement Format	Nonunit	Sequential Single Unit	Sequential Multiunit
CLOSE	C	C, G	A, C, G
CLOSE WITH LOCK	C, E	C, E, G	A, C, E, G
CLOSE WITH NO REWIND	C, H	B, C	A, B, C
CLOSE Unit	F	F, G	F, G
CLOSE Unit FOR REMOVAL	F	D, F, G	D, F, G

An explanation of the meaning of the letters within the preceding table is provided in the following sections.

### *A - Effect on Previous Units*

The results are listed as follows:

**Input and Input-Output Files-** All units in the file prior to the current unit are closed except those units that had been previously closed. If the current unit is not the last unit of the file, the following units in the file are not processed.

**Output Files-** All units in the file prior to the current unit are closed except those units that had been previously closed.

### *B - No Rewind of Current Unit*

For all files, the current unit is left in its current position.

### *C = Close File*

The results are listed as follows:

**Input and Input-Output Files-** When the file is positioned at its end and it has label records, the labels are processed. If the file is not positioned at its end, no label processing takes place. The file is closed and its associated record area is no longer available to the program.

If labels are specified but not present, or not specified and present, the behavior of the CLOSE statement is undefined.

**Output Files-** If the file has label records, they are processed. The file is closed and its associated record area is no longer available to the program. If labels are specified but not present, or not specified and present, the behavior of the CLOSE statement is undefined.

### *D - Unit Removal*

For **all files**, the **current** unit is rewound and logically removed from the run unit. A message is sent to the operator's console requesting the physical removal of the unit.

(As long as a unit is not physically removed, it can be accessed again by first issuing a CLOSE statement without the Unit phrase, followed by an OPEN statement.)

*E = File Lock*

For all files, the file is locked and cannot be opened again during the current execution of the run unit.

**F - Close Unit**

The **results** are listed as follows:

Input and Input-Output Files (Unit Media)- When the current unit is the last or *only* unit for the file, there is no unit swap, the current volume pointer remains unchanged, and the file position indicator is set to indicate that no next unit exists.

When another unit exists for the file, a unit swap occurs, the current volume pointer is updated to point to the next unit existing in the file, the standard beginning unit procedure is executed, and the file position indicator is set to one less than the number of the first record existing on the new current volume. If no data records exist for the current volume, another unit swap occurs.

Output Files (Reel Media) - The following operations take place:

The standard ending reel label procedure is executed.

- A reel swap takes place, and the current volume pointer is updated to point to the new reel.
- The standard beginning reel label procedure is executed
- The next WRITE statement executed for the file writes the data record to the next reel of the file.

Input Files, Input-Output Files, and Output Files (Nonunit Media) Execution of this statement is considered successful. The file remains in the OPEN mode, the file position indicator is unchanged, and no action takes place except as specified previously.

**G = Rewind**

For all files, the current unit is positioned at its physical beginning.

**H - Optional Phrases Ignored**

For all files, the CLOSE statement is executed as if none of the optional phrases were present.

## Explanation of CLOSE Statements for File Categories

The following table shows the results of executing each type of CLOSE:

CLOSE	A
CLOSE WITH LOCK	A B

The meaning of the letters within the preceding table is as follows:

### *A - Close File*

The results are listed as follows:

Input Files and Input-Output Files (Sequential Access Mode) - When the file is positioned at its end and it has label records, the labels are processed. If the file is not positioned at its end, no label processing takes place. The file is closed and its associated record area is no longer available to the program.

If labels are specified but not present, or not specified and present, the behavior of the CLOSE statement is undefined.

Input-Output Files (Random or Dynamic Access Mode) and Output Files in Any Access Mode - If the file has label records, they are processed. The file is closed and its associated record area is no longer available to the program. If labels are specified but not present, or not specified and present, the behavior of the CLOSE statement is undefined.

When the file does not have label records, it is just closed

### *B = File Lock*

For all files, the file is locked and cannot be opened again during the current program execution.

## COMMIT Statement

The COMMIT statement commits a transaction or subtransaction by writing its associated updates to disk and releases resources held by the program.

### **Format 1**

```
FREE ALL  
    [ON ERROR imperative-statement-1]  
    [NOT ON ERROR imperative-statement-2]  
    [END-FREE]
```

### **Format 2**

```
COMMIT [ data-name-1  
        literal-1 ]  
    [ ON ERROR      imperative-statement-1 ]  
    [ NOT ON ERROR  imperative-statement-2 ]  
    [END-COMMIT]
```

### *Syntax Rule*

Data-name-1 must be a numeric integer, literal-1 must be a positive integer.

### *General Rules*

1. The FREE ALL statement is equivalent to a COMMIT statement without either data-name-1 or literal-1. It commits the outermost transaction, including any nested subtransactions.
2. A COMMIT that specifies literal-1 or data-name-1 commits one or more transaction levels, as indicated by the value.
3. Execution of a COMMIT statement for a subtransaction causes that subtransaction to be merged with the next outermost transaction or subtransaction. The updates associated with the committed subtransaction (and any nested subtransactions) are written to disk, and all resources being held for it are released.
4. If the statement is not successful, execution continues according to the rules for exception handling.
5. If the statement is successful, the special-register RETURN-CODE is set to zero; if unsuccessful, the RETURN-CODE is set to 92.
6. If the statement is successful and the NOT ON ERROR phrase is included, execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the COMMIT statement. If the NOT ON ERROR phrase is not included, control passes to the end of the COMMIT statement.

## COMPUTE Statement

The COMPUTE statement evaluates an arithmetic expression and sets one or more data items equal to the result.

### **Format**

```
COMPUTE {data-name-1 [ROUNDED]}... = arithmetic-expression-1  
      [ ON SIZE ERROR imperative-statement-1 ]  
      [ NOT ON SIZE ERROR imperative-statement-2 ]  
      [ END-COMPUTE ]
```

### *Syntax Rule*

Data-name-1 must be numeric or numeric edited

### *General Rules*

1. The COMPUTE statement sets data-name-1 to the value of the arithmetic expression. More than one data-name-1 may be specified as a receiving data item for the value of the arithmetic expression. For additional details, refer to the sections titled "Arithmetic" and "Exception Handling Phrases" in Chapter 6.
2. The END-COMPUTE phrase delimits the scope of the COMPUTE statement.

## CONTINUE Statement

The CONTINUE statement is a no operation statement. The CONTINUE statement can be used anywhere a conditional statement or an imperative statement can.

Format

CONTINUE

The CONTINUE statement has no effect on the execution of a program.



## DELETE Statement

The DELETE statement removes a record from a relative or indexed file.

### Format

```
DELETE file-name-1 RECORD  
  
    [ INVALID KEY imperative-statement-1 ]  
  
    [ NOT INVALID KEY imperative-statement-2 ]  
  
    [ END-DELETE ]
```

### Syntax Rules

1. The INVALID KEY and NOT INVALID KEY phrases cannot be specified for a file in sequential ACCESS mode.
2. The INVALID KEY phrase must be specified for a file in RANDOM or DYNAMIC ACCESS mode that does not have an applicable USE procedure. However, if the Compiler option COMPAT74 is IO or YES, the INVALID KEY phrase need not be included, even if there is no applicable USE procedure.

### General Rules

1. File-name- I must be opened in 1-O or SHARED mode.
2. For files opened in sequential ACCESS mode, the last INPUT-OUTPUT statement executed for the file must have been a successfully executed READ statement (or READ WITH HOLD statement if SHARED OPEN mode). The DELETE statement removes the record accessed by that READ statement from the file.
3. For files opened in RANDOM or DYNAMIC ACCESS mode, the record identified by the file's key data item is removed. The key data item for relative files is the relative key data item. The key data item for indexed files is the prime record key data item. If the file does not contain the record specified by the key, the INVALID KEY condition exists.
4. After the successful execution of a DELETE statement, the deleted record can no longer be accessed.
5. The DELETE statement updates the associated FILE STATUS item, but does not affect the file position indicator, the record area of the fill, or the data item that appears in the DEPENDING ON phrase of the file's RECORD clause **fill**.

*Note: If the Compiler option COMPAT74 is YES or 10, and the file is in the RANDOM or DYNAMIC ACCESS mode, the key of reference becomes the primary key, and the current record pointer is positioned to the record immediately following the deleted one.*

6. If the DELETE is successful, and the NOT INVALID KEY phrase is not specified, control passes to the statement following the DELETE statement. If the NOT INVALID KEY phrase is specified, execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of

control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the DELETE statement.

7. If the DELETE is unsuccessful, execution continues according to the rules for exception handling.
8. The END-DELETE phrase delimits the scope of the DELETE statement.

## DISPLAY Statement

The DISPLAY statement transfers data to a workstation.

### **Format**

$$\text{DISPLAY } \left\{ \begin{array}{l} \text{literal-1} \\ \text{identifier-1} \end{array} \right\} \dots \left[ \text{UPON } \left\{ \begin{array}{l} \text{WORKSTATION} \\ \text{mnemonic-name-1} \end{array} \right\} \right] \\ [ \text{ WITH } \text{NO ADVANCING} ]$$

### **Syntax Rules**

1. Identifier-1 must have a usage of DISPLAY.
2. The Standard requires that if literal-1 is numeric, it must be an unsigned integer. As a Wang COBOL 85 extension, it can be signed and/or a noninteger.
3. Mnemonic-name-1 is specified as the WORKSTATION in the SPECIALNAMES paragraph.

### **General Rules**

1. The DISPLAY statement displays each operand on the workstation screen. Program execution is then suspended until the operator presses the Enter key. (If the workstation is open as a file at the time of the DISPLAY statement, the contents of the screen are saved before the message is displayed and are restored after the operator presses Enter.)
2. When a DISPLAY statement contains more than one operand, the size of the sending item is the sum of the sizes of the operands. The values of the operands are transferred in the sequence in which they are encountered, with no spaces inserted between the operands.
3. If a figurative constant is an operand, only a single instance of it is displayed.
4. Up to 18 rows of 79 characters each can be transferred for a total of 1,422 characters.
5. The WITH NO ADVANCING phrase is treated as a comment.

## DISPLAY AND READ Statement

The DISPLAY AND READ statement controls data displayed on and read from the workstation.

### Format

DISPLAY AND READ [ALTERED] record-name-1 ON file-name-1

$$\left[ \begin{array}{l} \left[ \text{ONLY} \right] \left\{ \begin{array}{l} \text{PFKEY} \\ \text{PFKEYS} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-1} \\ \text{integer-1} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-2} \\ \text{integer-2} \end{array} \right\} \dots \\ \left[ \text{ON} \right] \left\{ \begin{array}{l} \text{PFKEY} \\ \text{PFKEYS} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-3} \\ \text{integer-3} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-4} \\ \text{integer-4} \end{array} \right\} \dots \text{imperative-statement-1} \end{array} \right]$$

### Syntax Rules

1. Record-name-1 must be a Working-Storage data item with a usage of DISPLAYWS.
2. File-name-1 must specify a device of DISPLAY in the file-control entry.
3. Integer-1 and integer-2 can be from 1 to 32.
4. If a PF key is specified in the ON phrase, it must also be specified in the ONLY phrase.
5. The NO-MOD phrase can be specified only if ALTERED is specified.

### General Rules

1. First, the DISPLAY AND READ statement moves data from any data items specified in SOURCE clauses to the appropriate items in the record. Second, the record contents are displayed on the screen. The operator can now type in new values or modify the ones displayed. The operator response is then validated and the value moved to any data items specified in OBJECT clauses.
2. The data that is input is validated according to the specifications of the PICTURE and RANGE clauses for the item. (Alphanumeric and numeric edited data are converted to numeric before comparison with numeric items.) If a value is not compatible with these requirements, it is flagged as an error by having its FAC set to indicate blinking. (If a field that is already blinking passes this validation, while some other field does not, the FAC of the valid field may be set to nonblinking.)

In addition, the audio alarm is sounded, no data transfer occurs, and the screen record is displayed again. If a value meets these requirements, the data is transferred to the appropriate OBJECT field. When data is moved from numeric edited screen fields to numeric OBJECT fields, any currency signs, CR, DB, asterisks, and commas (or periods if the DECIMAL-POINT IS COMMA clause is in effect) are removed

3. A protected field is neither validated nor de-edited. Therefore, if the FAC of a numeric-edited field is changed to indicate a protected field, a subsequent DISPLAY AND READ could move invalid data to the OBJECT field. To avoid this, the subsequent DISPLAY AND READ may use the ALTERED phrase.

4. When ALTERED is specified, only those fields that have been modified by the user are manipulated.
5. Scaling characters or implied decimals in the PICTURE clauses of items are ignored in the screen display as well as in the transfer of data from the SOURCE field to the screen or from the screen to the OBJECT field. Furthermore, these characters do not display on the screen, nor are they replaced by pseudo-blanks. For purposes of data movement, a display field with such a PICTURE clause is treated as an integer field, unless decimal points are explicitly entered from the workstation by the user. If DECIMALPOINT IS COMMA is in effect, this rule applies to the comma.
6. If DECIMAL-POINT IS COMMA is in effect, and the PICTURE of a numeric-edited item does not contain a comma (that is, the item has an implicit decimal point), either a decimal point or a comma can be used in the input data to represent the decimal point. However, if the FAC of the numeric edited-field is set to allow only numeric input, only a period can be used to represent the decimal point.
7. If DECIMAL-POINT IS COMMA is in effect, and the PICTURE of a numeric-edited item contains a comma (that is, the item has an explicit decimal point), only a comma can be used in the input data to represent the decimal point. However, if the FAC of the numeric edited-field is set to allow only numeric input, no character can be used to represent the decimal point.
8. An operator response can either signal that data has been entered or indicate a special condition request, such as program termination, without entering data. The ON phrase identifies those PF keys used for special condition requests. When a key specified in the ON phrase is pressed, no data is transferred; rather, the associated imperative statement is executed, and the DISPLAY AND READ statement ends.
9. The ONLY phrase specifies which PF keys are considered valid responses for terminating input. The word ONLY is optional in this phrase. The following rules apply:
  - When ONLY is used, the only valid responses are the specified PF keys.
  - When ONLY is not used, but one or more PF keys are specified, the valid responses are one of the specified PF keys or the Enter key.
  - When neither ONLY nor a PF key is specified, the only valid response is the Enter key.
  - When either a valid PF key (that is not also specified in the ON phrase), or the Enter key (assuming it is a valid response) is selected and there are no input data errors, the data transfer occurs.
10. A total of 32 integer and data-name phrases can be used with the ON or the ONLY phrases.
11. The imperative statement of the NO-MOD phrase is executed only when there is no modification of any of the displayed fields. The NO-MOD phrase is ignored when a PF key that is specified in the ON phrase is pressed.

## DIVIDE Statement

The DIVIDE statement divides one item into another and sets one or more data items equal to the quotient and another data item equal to the remainder.

### Format 1

```
DIVIDE { identifier-1 } INTO { identifier-2 [ROUNDED] } ...  
      { literal-1 }  
[ ON SIZE ERROR imperative-statement-1 ]  
[ NOT ON SIZE ERROR imperative-statement-2 ]  
[ END-DIVIDE ]
```

### Format 2

```
DIVIDE { identifier-1 } INTO { identifier-2 }  
      { literal-1 }      { literal-2 }  
GIVING { identifier-3 [ROUNDED] } ...  
[ ON SIZE ERROR imperative-statement-1 ]  
[ NOT ON SIZE ERROR imperative-statement-2 ]  
[ END-DIVIDE ]
```

### Format 3

```
DIVIDE { identifier-1 } BY { identifier-2 }  
      { literal-1 }      { literal-2 }  
GIVING { identifier-3 [ROUNDED] } ...  
[ ON SIZE ERROR imperative-statement-1 ]  
[ NOT ON SIZE ERROR imperative-statement-2 ]  
[ END-DIVIDE ]
```

#### **Format 4**

DIVIDE { identifier-1 } INTO { identifier-2 }  
          { literal-1 }  
GIVING identifier-3 [ROUNDED]  
REMAINDER identifier-4  
[ ON SIZE ERROR imperative-statement-1 ]  
[ NOT ON SIZE ERROR imperative-statement-2 ]  
[ END-DIVIDE ]

#### **Format 5**

DIVIDE { identifier-1 } BY { identifier-2 }  
          { literal-1 }  
GIVING identifier-3 [ROUNDED]  
REMAINDER identifier-4  
[ ON SIZE ERROR imperative-statement-1 ]  
[ NOT ON SIZE ERROR imperative-statement-2 ]  
[ END-DIVIDE ]

#### *Syntax Rules*

1. All data-names must be numeric, except those associated with the GIVING or REMAINDER phrase, which can be numeric or numeric-edited
2. All literals must be numeric.
3. The composite of operands is formed by the receiving data items except the REMAINDER data item. It cannot contain more than 18 digits.

#### *General Rules*

##### *Format 1*

1. Literal-1 or identifier-1 is stored in a temporary data item. The value in this temporary item is then divided into identifier-2, the quotient becoming the new value of identifier-2. If more than one identifier-2 is specified, this action is repeated for each identifier, taking them in the left-to-right order in which they are written in the statement.

##### *Format 2*

2. Literal-1 or identifier-1 is divided into literal-2 or identifier-2, and the quotient is stored in each data item following GIVING.

##### *Format 3*

3. Literal-1 or identifier-1 is divided by literal-2 or identifier-2, and the result is stored in each data item following GIVING.

*Format 4*

4. Literal-1 or identifier-1 is divided by literal-2 or identifier-2, and the result is stored in each data item following GIVING. The remainder is calculated and stored in identifier-4. If identifier-4 is subscripted, the subscript is evaluated immediately before the remainder is stored.

*Format 5*

5. Literal-1 or identifier-1 is divided into literal-2 or identifier-2, and the result is stored in each data item following GIVING. The remainder is calculated and stored in identifier-4. If identifier-4 is subscripted, the subscript is evaluated immediately before the remainder is stored.

*Formats 4 and 5*

6. In COBOL, the remainder is defined as the result of subtracting the product of the quotient and the divisor from the dividend. If identifier-3 (the quotient) is numeric edited, the unedited quotient is used to calculate the remainder.
7. If **ROUNDED** is specified, the truncated quotient is used to calculate the remainder.
8. Decimal alignment and truncation (not rounding) is performed on the remainder before it is put into identifier-4.
9. If a size error occurs on the quotient, no remainder calculation is meaningful. Thus, the contents of both identifier-3 and identifier-4 are unchanged. If a size error occurs on the remainder, the contents of identifier-4 are unchanged. However, no indication of which item the size error occurred on is given, so the program must perform the analysis.

*All Formats*

10. If no size error occurs and the **NOT SIZE ERROR** phrase is not specified, program execution continues with a statement following the **DIVIDE**. If the **NOT SIZE ERROR** phrase is specified, execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the **DIVIDE** statement.
11. If a size error occurs, the contents of the affected data items are incorrect. Execution continues according to the rules for exception handling.
12. The **END-DIVIDE** phrase delimits the scope of the **DIVIDE** statement.



## ENTER Statement

The ENTER statement is treated as a comment.

### Format

```
ENTER language-name-1 [ routine-name-1 ] .
```

## EVALUATE Statement

The EVALUATE statement defines a multibranch, multijoin structure. Multiple conditions can be tested and different branches executed based on the results of these tests.

### Format

$$\begin{array}{l}
 \text{EVALUATE } \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \\ \text{expression-1} \\ \text{TRUE} \\ \text{FALSE} \end{array} \right\} \left[ \text{ALSO } \left\{ \begin{array}{l} \text{identifier-2} \\ \text{literal-2} \\ \text{expression-2} \\ \text{TRUE} \\ \text{FALSE} \end{array} \right\} \dots \right. \\
 \left. \left\{ \begin{array}{l} \text{WHEN} \\ \left\{ \begin{array}{l} \text{ANY} \\ \text{condition-1} \\ \text{TRUE} \\ \text{FALSE} \end{array} \right\} \\ \left[ \text{[NOT]} \left\{ \begin{array}{l} \text{identifier-3} \\ \text{literal-3} \\ \text{expression-3} \end{array} \right\} \left[ \text{THROUGH} \left\{ \begin{array}{l} \text{identifier-4} \\ \text{literal-4} \\ \text{expression-4} \end{array} \right\} \right] \right\} \end{array} \right\} \\
 \left[ \text{ALSO} \right. \\
 \left. \left\{ \begin{array}{l} \text{ANY} \\ \text{condition-2} \\ \text{TRUE} \\ \text{FALSE} \end{array} \right\} \\ \left[ \text{[NOT]} \left\{ \begin{array}{l} \text{identifier-5} \\ \text{literal-5} \\ \text{expression-5} \end{array} \right\} \left[ \text{THROUGH} \left\{ \begin{array}{l} \text{identifier-6} \\ \text{literal-6} \\ \text{expression-6} \end{array} \right\} \right] \right\} \right\} \\
 \text{imperative-statement-1} \left. \right\} \dots \\
 \left[ \text{WHEN OTHER imperative-statement-2} \right] \\
 \left[ \text{END-EVALUATE} \right]
 \end{array}$$

### Syntax Rules

1. Each operand that appears before any WHEN phrase is termed a *selection subject*. All the operands so specified in the statement are referred to collectively as the *set of selection subjects*. There is only one set of selection subjects.
2. Each operand that appears in a WHEN phrase is termed a *selection object*. All the operands in a single WHEN phrase are referred to as a *set of selection objects*. There are as many sets of selection objects as there are WHEN phrases.
3. The number of selection objects in each set of selection objects must be equal to the number of selection subjects.

4. Each selection object in a set must be consonant with the selection subject that has the corresponding ordinal position. The criteria for this agreement are
  1. <sup>r</sup>
    - A data-name, literal, or arithmetic expression that appears as, or in, a selection object must be able to be validly compared to the corresponding operand in the set of selection subjects.
    - If a condition or the words TRUE or FALSE appear as a selection object, the corresponding selection subject must be either a conditional expression or the words TRUE or FALSE.
    - The word ANY can correspond to any selection subject.
5. Two operands that are connected by a THROUGH phrase must belong to the same class. The two connected objects are considered a single selection object specifying a range.
6. Expression-3, expression-4, expression-5, and expression-6 must be arithmetic expressions.

### **General Rules**

1. First the EVALUATE statement evaluates each selection subject and selection object and assigns it a numeric or nonnumeric value, a range of numeric or nonnumeric values, or a truth value.

These values are determined as follows:

- A selection subject that is specified by identifier-1 or identifier-2, or a selection object (without the NOT or the THROUGH phrase) that is specified by identifier-3 or identifier-5, is assigned the value and class of the corresponding data item.
- A selection subject that is specified by literal-1 or literal-2, or a selection object (without the NOT or the THROUGH phrase), that is specified by literal-3 or literal-5, is assigned the value and class of the specified literal. If literal-3 or literal-5 is the figurative constant ZERO, the selection object is assigned the class of its corresponding selection subject.
- A selection subject in which expression-1 or expression-2 is an arithmetic expression, or a selection object without the NOT or the THROUGH phrase in which expression-3 or expression-4 is specified, is assigned the numeric value of the expression.
- A selection subject in which expression-1 or expression-2 is a conditional expression, or a selection object in which condition-1 or condition-2 is specified, is assigned a **truth value according** to the rules for evaluating conditional expressions.
- A selection subject or object specified by the words TRUE or FALSE is assigned the appropriate truth value.
- A selection object specified by the word ANY is not evaluated.
- If the THROUGH phrase without the NOT phrase is specified for a selection object, then the range includes all valid values for the selection subject that are

greater than or equal to the first operand and less than or equal to the second operand.

- If the NOT phrase is specified for a selection object, then all valid values that are either not equal to the value specified or not within the range specified, are assigned to the selection object. In other words, the set of values assigned is the complement of what would have been assigned if the NOT phrase had not been specified.
2. Second, the EVALUATE statement compares the values assigned to the selection subjects with those assigned to their corresponding selection objects to determine if any WHEN phrase satisfies all of the selection subjects. Each selection object in the first WHEN phrase is compared to its corresponding selection subject.

The comparisons are successful under the following conditions:

- If the items being compared have numeric or nonnumeric values, the comparison is successful if the value of the selection subject equals the value (or one out of the range of values) of the selection object.
- If the items being compared have truth values, the comparison is successful if the two items have the same truth value.
- If the selection object being compared is ANY, the comparison is always successful.
- If the comparison for every selection object within a set of selection objects is successful, their WHEN phrase is said to be selected.

These comparisons **are repeated for all** subsequent sets of selection objects, in the order in which the sets appear in the source program, until either a WHEN phrase is selected or all the sets of selection objects have been examined

3. When the comparison is completed and if a WHEN phrase was selected, execution continues with the associated imperative statement. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of the imperative statement, control is transferred to the end of the EVALUATE statement.
4. If no WHEN phrase was selected, but a WHEN OTHER phrase was specified, execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the EVALUATE statement. If no WHEN phrase was selected and a WHEN OTHER phrase was not specified, execution continues with the statement following the EVALUATE.

## EXIT Statement

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

### Format

EXIT

### *Syntax Rule*

The EXIT statement must appear in a sentence by itself, and this sentence must appear in a paragraph by itself.

### *General Rule*

The EXIT statement assigns a procedure-name to a point in the program.

## EXIT PROGRAM Statement

The EXIT PROGRAM statement marks the logical end of a called program.

### Format

EXIT PROGRAM

### *Syntax Rule*

If an EXIT PROGRAM statement is in a sequence of imperative statements in a sentence, it must be the last statement in that sequence.

### *General Rules*

1. When an EXIT PROGRAM statement is executed in a program that is not under the control of a calling program, it has no effect, and execution continues with the next executable statement in the program.
2. Execution of an EXIT PROGRAM statement in a called program returns control to the next executable statement following the CALL statement in the calling program. The program state of the calling program is not altered and is identical to what it was before the CALL statement was executed, except that the contents of the data items and files shared between the two programs might have been changed. The program state of the called program is not altered, except that the ends of the ranges of all PERFORM statements executed by that called program are considered to be reached.
3. If a called program has the initial attribute, the execution of an EXIT PROGRAM statement is equivalent to executing a CANCEL statement referencing the program.
4. An EXIT PROGRAM statement must not be executed in a GLOBAL declarative procedure, unless within a program called by a GLOBAL declarative procedure.

## FREE ALL Statement

The FREE ALL statement commits all uncommitted transactions and subtransactions and frees held resources. Refer to the section titled "COMMIT Statement" earlier in this chapter for details.

## GO TO Statement

The GO TO statement transfers control to a paragraph or section.

### *Format 1*

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

### *Format 2*

```
GO      TO { procedure-name-1 }... DEPENDING ON data-name-1
```

### *Syntax Rules*

1. Procedure-name- I is optional in format 1. In future revisions of the Standard, it will be mandatory.
2. The Standard requires that a paragraph referenced by an ALTER statement must consist of only a paragraph header followed by a format I GO TO statement.
3. If a format 1 GO TO statement has no procedure-name specified, the Standard requires that the statement be the only one in the paragraph.
4. The Standard requires that when a format 1 GO TO statement appears in a sequence of imperative statements in a sentence, it must be the last statement in that sequence. As a Wang extension, other statements can follow the GO TO statement.
5. Data-name-1 must be a numeric data item that is an integer.

### *General Rules*

1. Execution of a format 1 GO TO statement transfers control to procedurename-1.
2. A format I GO TO statement without a procedure-name- I must have an ALTER statement executed for it prior to the execution of the GO TO statement.
3. When a format 2 GO TO statement is executed, control is transferred to the procedure whose name is in the ordinal position designated by the current value of data-name-1. If its value is anything other than a positive or unsigned integer, or is greater than the number of procedure-names in the list, no transfer occurs and control passes to the statement following the GO TO.

## HOLD Statement

The HOLD statement reserves a file or a range of records in a file or reserves the exclusive right to claim these resources.

### Format

```
HOLD [LIST] RECORDS OF file-name-1
      { [FOR] { RETRIEVAL
                UPDATE
              } } [WITH KEYS { { INITIAL { literal-1
                                     { data-name-1
                                     ...CHARACTERS OF } { literal-2
                                                         { data-name-2
                                                         ... } } ... } } ...
      [ TIMEOUT OF { data-name-3 } [SECOND
                        integer-1   [SECONDS
                                     ]
                                     ]
      [ HOLDER-ID IN data-name-4 ]
      { imperative-statement-1 } ]
      { NEXT SENTENCE
      [ END-HOLD ]
```

### Syntax Rules

1. File-name-1 must be an indexed file or a sequential file opened in SHARED mode.
2. Data-name-1 must be numeric. Neither it nor literal-1 can contain a value that exceeds the length of data-name-2 (or literal-2) or the primary key of file-name-1.
3. Data-name-2 must have a usage of DISPLAY.
4. Literal-2 must be nonnumeric.
5. When the INITIAL phrase is not specified, the length of data-name-2 (or literal-2) cannot exceed the length of the prime record key of file-name-1.
6. The TIMEOUT phrase cannot be specified in the same HOLD statement as a LIST phrase, because the attempt to hold a list of resources is not made until the list is complete.
7. The TIMEOUT phrase is allowed with a HOLD statement that specifies more than one file-name or record group.
8. Integer-1 or data-name-3 must be less than 256.

## **General Rules**

1. Execution of a HOLD statement with the LIST phrase builds a list of files and/or a range of records from an indexed file. The LIST phrase specifies that the HOLD statement is part of a list of HOLD statements. Each HOLD statement in a list must have the LIST option, except for the last, which must not contain the LIST phrase. Upon execution of the last HOLD statement, an attempt to hold the entire list of requested resources is made.  
  
When RETRIEVAL or UPDATE is specified, it remains in effect until a different hold class or another RECORDS OF phrase is specified.
2. Multiple groups of records can be specified for a single file. In addition, both RETRIEVAL and UPDATE can be specified for different resources in the same file.
3. When a record group is not specified, all the records in the file are held. When a record group is specified, the records held are those identified by the following keys:
  - When INITIAL is not specified by the value of data-name-2 or literal-2
  - When INITIAL is specified by the first n characters of data-name-2 or literal-2, where n is the value of data-name-1 or literal-1
4. The HOLD statement updates the FILE STATUS item for the file or files being accessed.
5. When the TIMEOUT phrase is specified, the program waits integer-1 (or the value of data-name-4) seconds for the resources to be available. If the resources are not available in this time, control passes to either imperativestatement-1 or the next sentence. Specifying a value of zero seconds causes the timeout exit to be taken immediately if the resources cannot be held.  
  
If imperative-statement-1 is specified, execution continues with each statement in it. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-1, control is transferred to the end of the HOLD statement.
6. When the HOLDER-ID phrase is specified, if a timeout occurs, the PID of the user holding the resources is moved to data-name-4 in ASCII format.



## IF Statement

The IF statement evaluates a condition and takes subsequent action based on the truth value of that condition.

### **Format**

$$\begin{array}{c} \text{IF condition-1 THEN } \left\{ \begin{array}{l} \{\text{statement-1}\} \\ \text{NEXT SENTENCE} \end{array} \right\} \dots \\ \left[ \begin{array}{l} \text{ELSE } \left\{ \begin{array}{l} \{\text{statement-2}\}. \\ \text{NEXT SENTENCE} \end{array} \right\} \end{array} \right] \\ \text{[END-IF]} \end{array}$$

### **Syntax Rules**

1. Statement-1 and statement-2 can be either an imperative statement or a conditional statement.
2. The NEXT SENTENCE phrase can be omitted when it immediately precedes the period ending the sentence.
3. The END-IF phrase and the NEXT SENTENCE phrase cannot both be specified

### **General Rules**

1. The scope of an IF statement is terminated by an END-IF phrase at the same level of nesting, a separator period, or, when nested, by the ELSE phrase associated with an IF statement at a higher level of nesting.
2. When the condition is true and statement-1 is specified, control transfers to the first statement of statement-1 and continues with each statement in it. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of statement-1, control is transferred to the end of the IF statement. If NEXT SENTENCE is specified, control passes to the sentence following the IF statement.
3. When the condition is false and statement-2 is specified, control is transferred to the first statement of statement-2 and continues with each statement in it. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of statement-2, control is transferred to the end of the IF statement. If NEXT SENTENCE is specified, control passes to the sentence following the IF statement. If an ELSE phrase is not specified, control passes to the end of the IF statement.
4. Statement-1 and statement-2 can contain another IF statement. In this case, the IF statements are said to be nested. Nested IF statements are considered to be paired IF, ELSE, and END-IF combinations, proceeding from left to right. Thus, any ELSE or END-IF phrase is considered to be associated with the immediately preceding IF that has not already been paired with an ELSE or END-IF phrase.

## INITIALIZE Statement

The INITIALIZE statement assigns values to data items. This provides a way of initializing file items that cannot have a VALUE IS clause.

### Format

```
INITIALIZE {data-name-1}...  
  
[ REPLACING { { ALPHABETIC  
                  ALPHANUMERIC  
                  NUMERIC  
                  ALPHANUMERIC-EDITED  
                  NUMERIC-EDITED } } ]  
  
DATA BY { { data-name-2 } } ... ]  
          { literal-1 } }
```

### Syntax Rules

1. Literal- I or data-name-2 is the sending item. Data-name-1 is the receiving item. Each category of item specified in the REPLACING phrase must be such that it would be a legal receiving operand in a MOVE statement in which data-name-2 or literal-1 was the sending operand
2. A given category cannot appear more than once in a REPLACING phrase.
3. The description of data-name-1 as well as any items subordinate to it cannot contain an OCCURS clause with a DEPENDING phrase or a RENAME clause.
4. An index data item cannot be an operand of an INITIALIZE statement.

### General Rules

1. Each keyword following REPLACING specifies a data category.
2. Regardless of whether data-name-1 is an elementary item or a group item, all INITIALIZE operations are performed as if a series of MOVE statements were executed, each of which has an elementary item as its receiving operand and data-name-2 or literal-1 as the sending operand
3. The data items are set to their values in the same left-to-right order as they appear in the statement. When a group item is referenced, the affected elementary items are initialized in the order in which they are defined in the group.
4. If the REPLACING phrase is specified and data-name-1 is a group item, an elementary item in the group is initialized only if it belongs to the category specified in the REPLACING phrase. If data-name-1 is an elementary item, the item is initialized *only* if it belongs to the category specified in the REPLACING phrase.
5. If the REPLACING phrase is not specified, alphabetic, alphanumeric, and alphanumeric edited data items are set to spaces, and numeric and numeric edited data items are set to zero.

6. Index data items and elementary FILLER data items are not affected by the INITIALIZE statement.
7. If data-name-1 overlaps any of the storage area of data-name-2, the result of the INITIALIZE statement is not predictable.
8. A data item with a REDEFINES clause that is subordinate to a receiving item, or any item(s) subordinate to such an item, is not affected by the INITIALIZE statement. However, a receiving item can itself have a REDEFINES clause or be subordinate to a data item with a REDEFINES clause.

## INSPECT Statement

The INSPECT statement counts and/or replaces single characters or groups of characters in a data item.

### Format 1

```
INSPECT identifier-1 TALLYING {
    {
        {CHARACTERS} [ {BEFORE} INITIAL {identifier-4} ] ...
                    [ {AFTER}  INITIAL {literal-2}  ] ...
    }
    {
        {ALL}        { {identifier-3} [ {BEFORE} INITIAL {identifier-4} ] ... } ...
        {LEADING}    { {literal-1}   [ {AFTER}  INITIAL {literal-2}  ] ... } ...
    }
} ...
```

### Format 2

INSPECT identifier-1 REPLACING

```
{
    {identifier-5} [ {BEFORE} {identifier-4} ]
{CHARACTERS BY}  { } { } INITIAL { } ...
{
    {literal-3}  [ {AFTER} {literal-2} ]
}
{
    {ALL}        { {identifier-3} {identifier-5} [ {BEFORE} {identifier-4} ] }
    {LEADING}    { } BY { } { } INITIAL { } ... } ...
    {FIRST}     { {literal-1} {literal-3} [ {AFTER} {literal-2} ] }
}
```

### Format 3

INSPECT identifier-1 TALLYING { identifier-2 FOR

$$\left\{ \begin{array}{l} \text{CHARACTERS} \left[ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right] \text{INITIAL} \left\{ \begin{array}{l} \text{identifier-4} \\ \text{literal-2} \end{array} \right\} \dots \\ \left\{ \begin{array}{l} \text{ALL} \\ \text{LEADING} \end{array} \right\} \left\{ \left\{ \text{identifier-3} \right\} \left[ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right] \text{INITIAL} \left\{ \begin{array}{l} \text{identifier-4} \\ \text{literal-2} \end{array} \right\} \dots \right\} \dots \end{array} \right\} \dots$$

REPLACING

INSPECT identifier-1 REPLACING

$$\left\{ \begin{array}{l} \text{CHARACTERS BY} \left\{ \begin{array}{l} \text{identifier-5} \\ \text{literal-3} \end{array} \right\} \left[ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right] \text{INITIAL} \left\{ \begin{array}{l} \text{identifier-4} \\ \text{literal-2} \end{array} \right\} \dots \\ \left\{ \begin{array}{l} \text{ALL} \\ \text{LEADING} \\ \text{FIRST} \end{array} \right\} \left\{ \left\{ \text{identifier-3} \right\} \text{BY} \left\{ \begin{array}{l} \text{identifier-5} \\ \text{literal-3} \end{array} \right\} \left[ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right] \text{INITIAL} \left\{ \begin{array}{l} \text{identifier-4} \\ \text{literal-2} \end{array} \right\} \dots \right\} \dots \end{array} \right\} \dots$$

### Format 4

INSPECT identifier-1 CONVERTING

$$\left\{ \begin{array}{l} \text{identifier-6} \\ \text{literal-4} \end{array} \right\} \text{TO} \left\{ \begin{array}{l} \text{identifier-7} \\ \text{literal-5} \end{array} \right\}$$

$$\left\{ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right\} \text{INITIAL} \left\{ \begin{array}{l} \text{identifier-4} \\ \text{literal-2} \end{array} \right\} \dots$$

### Syntax Rules

#### Syntax Rules

#### All Formats

1. All data items except data-name-1 and data-name-2 must be elementary items with a usage of DISPLAY.
2. Data-name-1 must be a group item or an elementary item with a usage of DISPLAY.
3. Each literal must be nonnumeric and cannot be a figurative constant beginning with the word ALL. When literal-1, literal-2, or literal-4 is a figurative constant, its length is one character.
4. No more than one BEFORE phrase and one AFTER phrase can be specified for any given ALL, LEADING, CHARACTERS, FIRST, or CONVERTING phrase.

#### Formats 1 and 3

5. Data-name-2 must be numeric.

#### Formats 2 and 3

6. The size of literal-3 or data-name-5 must be equal to the size of literal-1 or data-name-3. If literal-3 is a figurative constant, its size is taken as equal to literal-1 or data-name-3.

7. If the CHARACTERS phrase is specified, literal-2, literal-3, data-name-4, or data-name-5 must be one character in length.

#### *Format 4*

8. The size of literal-5 or data-name-7 must be equal to the size of literal-4 or data-name-6. If literal-5 is a figurative constant, its size is taken as equal to the size of literal-4 or data-name-6.
9. The same character must not appear more than once in either literal-4 or data-name-6.

#### *General Rules*

##### *All Formats*

1. An INSPECT operation includes the establishment of boundaries for the BEFORE and AFTER phrase, a comparison cycle, and the mechanism for tallying and/or replacing. Inspection begins with the leftmost character of data-name-1 and proceeds to the rightmost character as described in step 3 of the General Rules.
2. Data-name-1, data-name-3, data-name-4, data-name-5, data-name-6, and data-name-7 are treated as follows:
  - If the item is alphabetic or alphanumeric, the contents are treated as a character-string.
  - If the item is alphanumeric edited, numeric edited, or unsigned numeric, it is treated as if it were redefined as alphanumeric.
  - If the item is signed numeric, it is treated as though it were first moved to an unsigned numeric data item of equal length (excluding any separate sign position). The procedure for an unsigned numeric item is then applied. If data-name-1 is signed numeric, the original value of the sign is retained upon completion of the INSPECT statement.
3. Subscripting associated with any data-name is evaluated only once as the first operation in the execution of the INSPECT statement.
4. In the General Rules 5 through 19, references to literal-1, literal-2, literal-3, literal-4, or literal-5 also apply to the corresponding data items (data-name-3, data-name-4, data-name-5, data-name-6, and data-name-7), respectively.

##### *Formats 1 and 2*

5. During the inspection of data-name-1, each properly matched occurrence of literal-1 is tallied (format 1) or replaced by literal-3 (format 2).
6. The comparison operation to determine the occurrence of literal-1 to be tallied or replaced is as follows:
  - The operands of the TALLYING or REPLACING phrase are considered in the left-to-right order in which they are specified in the INSPECT statement. The first literal-1 is compared to an equal number of contiguous characters in data-name-1, starting with the leftmost eligible character. For literal-1 to match that portion of data-name-1, the two strings must be identical. If they are, and

neither LEADING or FIRST is specified, they match. If LEADING applies to literal-1, that occurrence must be a leading one in order to match. If FIRST applies to literal-1, that occurrence must be the first one in data-name-1 to match

- If no match occurs, the comparison is repeated with each successive literal-1 until either a match is found or there is no next successive literal-1. In the latter case, the character of data-name-1 that is immediately to the right of the leftmost character considered in the comparison cycle now becomes the leftmost character, and the comparison cycle begins again with the first literal-1.
- If a match occurs, tallying or replacing takes place. The character in data-name-1 immediately to the right of the rightmost character that participated in the successful match is now considered to be the leftmost eligible character of data-name-1, and the comparison cycle starts again with the first literal-1.
- When the rightmost character of data name-1 has participated in a match or has been considered as the leftmost character, the inspection terminates.
- If the CHARACTERS phrase is specified, an implied 1-character operand participates in the cycle described in General Rule 6, but no comparison actually takes place. This implied character is considered to match every character of data-name-1.

7. If the BEFORE or AFTER phrase is specified, the number of characters of data-name-1 that are considered for comparison is altered.

*Note: In this rule, anything that applies to literal-1 also applies to the implied operand of the CHARACTERS phrase.*

If the BEFORE phrase is specified literal-1 participates only in those comparison cycles involving that part of data-name-1 from its leftmost character up to, but not including, the first occurrence of literal-2 in the item. (If there is no occurrence of literal-2 in data-name-1, the comparison operation is the same as if the BEFORE phrase had not been specified.)

The location of the first occurrence of literal-2 is determined before the first cycle of comparison is begun. If, on any comparison cycle, literal-1 is not eligible to participate, it is considered not to match.

If the AFTER phrase is specified, literal-1 participates only in those comparison cycles involving that part of data-name-1 from the character immediately to the right of the rightmost character of the first occurrence of literal-2 to the rightmost character of data-name-1.

The location of this first occurrence of literal-2 is determined before the first cycle of comparison is begun. If, on any comparison cycle, literal-1 is not eligible to participate, it is considered not to match.

If there is no occurrence of literal-2 in data-name-1, literal-1 is not eligible to participate in the comparison operation.

#### *Format 1*

8. The words ALL and LEADING apply to each succeeding literal-1 until a different word appears.
9. The contents of data-name-2 are not initialized by the INSPECT statement.

10. When the ALL phrase is specified, data-name-2 is incremented by one for each occurrence of literal-1 that matches in data-name-1.
11. When the LEADING phrase is specified, data-name-2 is incremented by one for the first and each subsequent contiguous occurrence of literal-1 that matches in data-name-1, provided that the **first** such occurrence of literal-1 is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate. .
12. If the CHARACTERS phrase is specified, data-name-2 is incremented by one for each character in data-name-1.
13. If data-name-1, data-name-3, or data-name-4 occupies any part of the storage area of data-name-2, the result of the INSPECT statement is unpredictable, even if these data items are in the same data description entry.

*Format 2*

14. The words ALL, LEADING, and FIRST apply to each succeeding BY phrase until a different word appears.
15. If the CHARACTERS phrase is specified, each character in data-name-1 is replaced by literal-3.
  - If ALL is specified, each occurrence of literal-1 in data-name-1 is replaced by literal-3.
  - If LEADING is specified, the **first and** each successive contiguous occurrence of literal-1 in data-name-1 is replaced by literal-3, provided that the first occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.
  - If FIRST is specified, the leftmost occurrence of literal-1 in data-name-1 is replaced by literal-3. This rule applies to each successive specification of the FIRST phrase, regardless of the value of literal-1.
16. If data-name-3, data-name-4, or data-name-5 occupies any part of the storage area of data-name-1, the result of the INSPECT statement is unpredictable, even if these data items are in the same data description entry.

*Format 3*

17. A format 3 INSPECT statement is considered the same as two successive INSPECT statements specifying the same data-name-1. One of the statements is interpreted as a format 1 INSPECT statement with TALLYING phrases, while the other is interpreted as a format 2 INSPECT statement with REPLACING phrases. Subscripting associated with any data-name in the format 2 statement is evaluated only once, before the format 1 statement is executed.

*Format 4*

18. A format 4 INSPECT statement is the same as a format 2 INSPECT statement that specifies the same data-name-1 and that has a series of ALL phrases, one for each character of literal-4. Each of these ALL phrases references as literal-1, a single character of literal-4, and references as literal-3, the corresponding single character of literal-5. Correspondence between the characters of literal-4 and the characters of literal-5 is by ordinal position in the data item.
19. When data-name-4, data-name-6, or data-name-7 occupy any part of the storage area of data-name-1, the result of the INSPECT statement is unpredictable, even if these data items are in the same data description entry.



## MERGE Statement

The MERGE statement combines two or more ordered files based on the values of one or more keys. During this process, the records in the files being merged are made available, in merged order, to a file or an output procedure:

### Format

```
MERGE file-name-1 { ON { ASCENDING } KEY {data-name-1} ... } ...  
[ COLLATING SEQUENCE IS alphabet-name-1 ]  
USING file-name-2 { file-name-3 } ...  
{ GIVING { file-name-4 } ...  
  OUTPUT PROCEDURE IS procedure-name-1 [THROUGH procedure-name-2] }
```

### Syntax Rules

1. A MERGE statement cannot appear in a Declarative section.
2. File-name-1 must be a sort-merge file.
3. If file-name-1 has fixed-length records, the size of the records in file-name-2 and file-name-3 cannot be larger than the largest record type in file-name-1. If file-name-1 contains variable-length records, the size of the records in filename-2 and file-name-3 cannot be smaller than the smallest record type nor greater than the largest record type in file-name-1.
4. Data-name-1 is the name of a key data item. Key data items are those on which the merge is based. Up to eight key data items can be specified, and they can be qualified.

Key data items are subject to the following rules:

- A key data item must be in a record of file-name-1. It cannot have an OCCURS clause in its description nor be subordinate to an item that contains an OCCURS clause. If it is a group item, it cannot contain a variable occurrence data item.
  - If file-name-1 has more than one record description, key data items need be described in only one of the record descriptions. The same character positions are taken as the key in all the record types of the file.
  - If file-name-1 contains variable-length records, all key data items must be contained in the first x character positions of the record, where x equals the size of the smallest record of file-name-1.
5. File-name-2, file-name-3, and file-name-4 cannot be sort-merge files.

6. If file-name-4 is an indexed file, the first data-name-1 must be ASCENDING, and data-name-1 must occupy the same character positions in its record as the prime record key for file-name-4.
7. If file-name-4 contains fixed-length records, the size of the records contained in file-name-1 must not be greater than the largest record type in file-name-4. If file-name-4 contains variable-length records, the size of the records in filename-1 cannot be smaller than the smallest record type or greater than the largest record type in file-name-4.
8. A file-name cannot appear more than once in a given MERGE statement.
9. If a file is on a multiple file reel, no other file in the MERGE statement can be on that reel.
10. No two file-names can be specified in the identical SAME AREA, SAME SORT AREA, or SAME SORT-MERGE AREA clause.
11. The only file-names that can be specified in the identical SAME RECORD AREA clause are those in the GIVING phrase.

### **General Rules**

1. The MERGE statement merges all the records in file-name-2 and file-name-3 and then transfers them to file-name-1. Neither file-name-2 nor file-name-3 can be open when the MERGE statement is executed.
2. The result of a merge is not correct unless the records in all of the files being merged are in order according to the key data items specified in the ASCENDING or DESCENDING phrases.
3. If ASCENDING is specified, the merged sequence is from the lowest values of the key data items to the highest. If DESCENDING is specified; the merged sequence is from the highest values of the key data items to the lowest. These comparisons are done according to the rules for comparison of operands in a RELATION condition.
4. The MERGE statement performs the following actions for each file to be merged:
  - An implicit OPEN INPUT statement is executed for the file. If an output procedure is specified, this implicit opening is performed before control passes to that procedure.
  - The records of the file are obtained and released to the MERGE operation by an implicit READ statement with the NEXT and AT END phrases.
  - An implicit CLOSE statement, without any optional phrases is executed *for each file. If an output* procedure is specified, this closing is not performed *until control leaves that procedure.*
  - These implicit functions are performed such that any associated USE AFTER EXCEPTION procedure is executed.
5. If file-name-1 contains only fixed-length records, any record in file-name-2 or file-name-3 that contains fewer character positions than that record size is space-filled on the right when the record is transferred to file-name-1.

6. The data-names following KEY are taken, from left to right, as having decreasing significance. The leftmost data-name is the major or most significant key, the next data-name is the next most significant key, and so on. This is without regard as to how they are divided among KEY phrases. In other words, the two phrases

ASCENDING KEY-1, KEY-2, KEY-3

ASCENDING KEY-1, ASCENDING KEY-2, DESCENDING KEY-3

both specify KEY-1 as the most significant key, KEY-3 as the least significant key, and KEY-2 as the one in between.

7. If **all** of the key data items of one record are equal to all the corresponding key data items of one or more other records, the records are returned in the order in which their files are specified in the MERGE statement: Further-more, all such equal records in one file are returned before any records of the other files.
8. If the COLLATING SEQUENCE phrase is specified, that collating sequence is used in comparison of nonnumeric key data items. If not, the program collating sequence is used.
9. If the GIVING phrase is specified, the merged records are written to filename-4. At the start of execution of the MERGE statement, this file cannot be open. For each file in this phrase
- An implicit OPEN OUTPUT statement is executed, thereby creating the file.
  - The merged records are written to the file by an implicit WRITE statement without any optional phrases. If file-name-4 contains only fixedlength records, any record in file-name-1 that contains fewer character positions is space filled on the right when that record is written to filename-4.
  - For relative files, the relative key data item for the first record contains the value 1; for the second record, it contains the value 2, and so on. After completion of the MERGE statement, the relative key data item has the relative record number of the last record returned to the file.
  - An implicit CLOSE statement without any optional phrases is executed for the file.
  - These implicit functions are performed such that any associated USE AFTER EXCEPTION procedure is executed. However, such a USE procedure must not execute any statement manipulating file-name-4 or accessing its record area.
  - If there is an attempt to write more records than the file can hold, and a USE AFTER EXCEPTION procedure is specified for the file, it is executed. If control is returned from the USE procedure, or if such a procedure is not specified, an implicit CLOSE statement without any optional phrases is executed for the file.
10. An output procedure can consist of any statements that select, modify, or copy the merged records from file-name-1 that are made available to it, one at a time, by its RETURN statement.
11. The range of the output procedure includes not only the statements. in the specified paragraphs or sections, but also any statements that are executed as the result of

executing any CALL, EXIT, GO TO, and PERFORM statements in these procedures. In addition, the range includes any statements executed in associated declarative procedures. This range must not cause the execution of any MERGE, RELEASE, or SORT statement

12. If an output procedure is specified, control passes to the output procedure during execution of the MERGE statement. When control passes the last statement in the output procedure, the MERGE statement ends. Before entering the output procedure, the merge operation is at a point at which it can produce the next record in merged order when requested by a RETURN statement in the output procedure.
13. During execution of the output procedure (including execution of any USE AFTER EXCEPTION procedure that is implicitly invoked while executing the MERGE statement), no statement can be executed that manipulates filename-2 and file-name-3 or their record areas.
14. During execution of any USE AFTER EXCEPTION procedure that is invoked while executing the MERGE statement, no statement can be executed that manipulates file-name-4 or its record area.

## MOVE Statement

The MOVE statement transfers data to one or more data items.

### **Format 1**

MOVE  $\left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\}$  TO {identifier-2}...

### **Format 2**

MOVE CORRESPONDING identifier-1 TO identifier-2

### **Format 3**

MOVE WITH CONVERSION identifier-1 TO identifier-2  
[ ON ERROR imperative-statement-1 ]  
[ END-MOVE ]

### **Format 4**

MOVE figurative-constant-1 TO FAC OF identifier-2

### **Format 5**

MOVE FAC OF identifier-1 TO identifier-2

### **Format 6**

MOVE identifier-1 TO ORDER-AREA OF record-name-2

### **Format 7**

MOVE ORDER-AREA OF record-name-1 TO identifier-2

## Syntax Rules

1. Literal-1, figurative-constant-1, identifier-1, or record-name-1 is the sending item. The data item referenced by identifier-2 or record-name-2 is the receiving item.
2. An index data item cannot be an operand of a MOVE statement.
3. In a MOVE WITH CONVERSION, identifier-2 must be numeric.
4. The FAC OF phrase can apply to a single display item or to a display item with an OCCURS clause.
5. Figurative-constant-1 must be defined as a 1-byte item.
6. Record-name-1 and record-name-2 must have a usage of DISPLAY-WS or be a record of a file with a device type of DISPLAY.
7. In a format 2 MOVE, all data-names must be group items.

## General Rules

### *Formats 1 and 2*

1. The sending item is moved to each receiving item in the order in which the receiving items are specified in the statement.
2. Any subscripting or length evaluation associated with a sending item is evaluated only once before data is moved to the first receiving item.
3. Any length evaluation or subscripting associated with a receiving item is evaluated immediately before the data is moved to it.
4. A numeric literal belongs to the numeric category; nonnumeric literals belong to the alphanumeric category.
5. The figurative-constant ZERO, when moved to a numeric or numeric edited item, belongs to the numeric category; in all other cases, it belongs to the alphanumeric category.
6. The figurative-constant SPACE belongs to the alphabetic category. All other figurative-constants belong to the alphanumeric category.
7. Any move in which the receiving item is an elementary item and the sending item is either a literal or an elementary item is an elementary move.
8. A move that is not an elementary move is a group move. A group move is strictly a byte to byte move, with no conversion of data whatsoever taking place. This means that the receiving area is filled without consideration for the individual elementary or group items in either the sending or receiving item, unless the item in question is a variable occurrence data item. Consequently, group moves should be made judiciously to ensure that the contents of data items are kept intact. Refer to the section titled "OCCURS Clause" in Chapter 5 for further information on variable occurrence data items.

*Note: The following rules apply only to elementary moves.*

9. In an elementary move, any necessary conversion of data from one form of internal representation to another takes place, along with any necessary editing or de-editing.

In addition, alignment by decimal points and any required zero filling take place, except where zeros are replaced by editing. For an elementary move

- A figurative constant other than ZERO, an alphanumeric edited data item, or an alphabetic data item cannot be moved to a numeric or numeric edited data item.
  - A numeric data item or literal, a numeric edited data item, or the figurative constant ZERO cannot be moved to an alphabetic data item.
  - A noninteger numeric data item or literal cannot be moved to an alphanumeric or alphanumeric edited item.
  - All other elementary moves are valid.
10. If the sending item is numeric and its PICTURE contains P's, all digit positions so designated are considered to contain a zero and are counted in the size of the sending item.
  11. If the sending item is signed numeric, the operational sign is not moved even if it occupies a separate character position. In the latter case, the size of the sending item is considered to be one less than its actual size.
  12. If the sending item is numeric edited, no de-editing takes place.
  13. If the receiving item is numeric or numeric edited and the sending item is alphabetic, data is moved as if the sending item were an unsigned integer. If the sending item is numeric edited, implicit de-editing is done to determine the unedited value, which can be signed. This unedited value is then moved to the receiving item.
  14. If the receiving item is signed numeric, the sign of the sending item is placed in the receiving item. Conversion of the representation of the sign takes place as necessary. If the sending item is unsigned, a positive sign is put into the receiving item.
  15. If the receiving item is unsigned numeric, the absolute value of the sending item is moved and no operational sign is generated for the receiving item. This means that a negative value so moved becomes positive.
  16. If the receiving item is alphabetic, alphanumeric, or alphanumeric edited, justification and any necessary space filling takes place according to the standard alignment rules.

17. In a format 2 MOVE CORRESPONDING, items in identifier-1 that have corresponding items in identifier-2 are moved to those items according to the rules described in the previous General Rules. The result is the same as if each pair of corresponding identifiers appeared in a separate MOVE statement.

The following table summarizes the valid format 1 and format 2 elementary moves.

**Valid Move Statements**

Category of Sending Item	Category of Receiving Item		
	Alphabetic	Alphanumeric Edited Alphanumeric	Numeric Integer Numeric Noninteger Numeric Edited
Alphabetic	Yes	Yes	No
Alphanumeric	Yes	Yes	Yes
Alphanumeric Edited	Yes	Yes	No
Numeric Integer	No	Yes	Yes
Numeric Noninteger	No	Yes	Yes
Numeric Edited	No	Yes	Yes

Format 3

18. A MOVE WITH CONVERSION converts a character-string that contains a representation of a number into a numerical value that can be used for computation.
19. Identifier-1 can be either alphanumeric or numeric edited, but it cannot contain more than 16 characters. It can have leading and trailing blanks but not embedded ones. The digits can be immediately preceded or followed by a plus or minus sign. If neither is specified, the plus sign is assumed. There can be a decimal point anywhere in the number. If none is present, a decimal point is assumed to be positioned to the right of the last digit. The number is aligned in identifier-2 by this explicit or implicit decimal point. Other than this optional decimal point, there cannot be any embedded nonnumeric characters in the number, in particular, it cannot contain commas. If the DECIMAL-POINT IS COMMA clause is in effect, this rule applies to the comma, and the number cannot contain any embedded decimal points.
20. If the ON ERROR phrase is specified and identifier-1 does not contain data conforming to the preceding rules, truncation occurs on either end of the item when it is moved into the receiving field, the truncated value is left in identifier-2, and execution continues with each statement in imperative-statement-1. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-1, control is transferred to the end of the MOVE statement.



*Formats 4 through 7*

21. For a single display-item, the FAC can be processed by using a FAC OF display-item. For a display-item that has an OCCURS clause, the FAC of the display-item must be written with subscripts indicating the particular display-item intended.

*Note: The hexadecimal characters used for FACs and for order area control are listed in Appendix C, in the section titled "Gist of Field Attribute Characters," and in Appendix D.*

## 1. MULTIPLY Statement

The MULTIPLY statement multiplies two items and sets one or more data items equal to the result.

### **Format 1**

```
MULTIPLY { identifier-1 }  
          { literal-1   } } BY { identifier-2 [ROUNDED] } ...  
  
[ ON SIZE ERROR imperative-statement-1 ]  
[ NOT ON SIZE ERROR imperative-statement-2 ]  
[ END-MULTIPLY ]
```

### **Format 2**

```
MULTIPLY { identifier-1 } BY { identifier-2 }  
          { literal-1   }      { literal-2   }  
  
GIVING { identifier-3 [ROUNDED] } ...  
[ ON SIZE ERROR imperative-statement-1 ]  
[ NOT ON SIZE ERROR imperative-statement-2 ]  
[ END-MULTIPLY ]
```

### **Syntax Rules**

1. Each operand must be numeric, except that in format 2, an operand following GIVING can be numeric or numeric edited.
2. Each literal must be numeric.
3. The composite of operands must not contain more than 18 digits.

## **General Rules**

### **Both Formats**

1. If no size error occurs and the NOT SIZE ERROR phrase is not specified, program execution continues with the statement following the MULTIPLY. If the NOT SIZE ERROR phrase is specified execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the MULTIPLY statement.
2. If a size error occurs, the contents of the affected data items are incorrect. Execution continues according to the rules for exception handling.

### **Format 1**

- 3 Literal- I or identifier-1 is stored in a temporary data item. This temporary data item is then multiplied by each identifier-2, with the product becoming the new value of that data item.

### **Format 2**

- 4 Literal-1 or identifier-1 is multiplied by literal-2 or identifier-2. The product is put into each data item following GIVING.

## OPEN Statement

The OPEN statement makes a file available to the program.

### Format

$$\text{OPEN } \left\{ \begin{array}{l} \text{INPUT} \\ \text{OUTPUT} \\ \text{EXTEND} \\ \text{I-O} \\ \text{SHARED} \\ \text{SPECIAL-INPUT} \end{array} \right\} \left\{ \text{file-name-1 } [\text{WITH NO REWIND}] \right\} \dots$$

### Syntax Rules

1. The files referenced in the OPEN statement need not all have the same organization or access.
2. The EXTEND phrase can be used only with a file in sequential ACCESS mode.
3. The WITH NO REWIND clause can only be specified with the INPUT or OUTPUT phrases applied to files with SEQUENTIAL organization.
4. The SPECIAL-INPUT phrase can be used only with a file whose organization is indexed, has an ACCESS mode of sequential, and has at least one alternate key. For all other files, SPECIAL-INPUT is treated as INPUT, and a diagnostic message is issued
5. The SHARED phrase can be used only for a file whose device is DISK.
6. The I-O phrase can be used only for a file whose device is DISK or DISPLAY.
7. A file whose device is DISPLAY can only be opened in the I-O mode.

### General Rules

1. When a file is opened, it is associated with the file-name used in the program and with the file connector. In addition, the record area of the file is made available to the program. (The OPEN statement does not obtain the first data record, however.) If the file is external, only one record area is associated with the file connector for the entire run unit, even though the file may be specified in more than one program.
2. An OPEN statement must be successfully executed prior to the execution of any other INPUT-OUTPUT statement. Except for OPEN, the only statements that can be executed on a closed file are the MERGE and SORT statements and then only when the file is specified in a USING or GIVING phrase.
3. An OPEN statement cannot be issued for an open file.
4. The particular INPUT-OUTPUT statements that can be used with a file depend on the mode in which the file is opened. The following table indicates which statements are valid for the different ACCESS and OPEN modes and file organizations. With the ACCESS mode given on the left and the OPEN mode given at the top of the column, an S in a box indicates the statement can be used for a file whose

organization is sequential, an R indicates it can be used for a Relative file, and X indicates it can be used for an Indexed file.

**Permissible I-O Statements for Open Files**

5. A file can be closed and reopened in any mode during the execution of the run unit.
6. When a file is opened in INPUT mode
  - The current volume pointer is set to indicate the first or only unit of the file.
  - For a Sequential or Relative file, the file position indicator is set to 0. For an Indexed file, the file position indicator is set to the characters with the lowest position in the collating sequence associated with the file, and the prime record key is established as the key of reference.
  - If no records exist in the file, the file position indicator is set so that the next READ statement for the file results in the AT END condition
  - If the file is an optional file and is not present, the file position indicator is set to indicate this condition
  - If the file has label records, they are checked according to the conventions for input labels. If label records are specified but not present, or not specified and present, the program is in error.
7. When a file is opened in OUTPUT mode
  - If the file does not exist, it is created and the current volume pointer set to indicate the new unit. If a file with the same name does exist, the operator is asked whether or not the existing file should be deleted.
  - If the file has label records, they are written according to the conventions for output labels.
  - Only the last file on a multiple file tape can be opened for output.
8. When a file is opened in I-O mode
  - The device specified in the FILE-CONTROL entry of the file must support both input and output operations.
  - For a Sequential or Relative file, the file position indicator is set to 0. For an Indexed file, the file position indicator is set to the characters with the lowest position in the collating sequence associated with the file, and the prime record key is established as the key of reference.
  - If an optional file is not present, it is created. The current volume pointer is set to indicate the new unit.
  - If the file has label records, they are checked in accordance with the conventions for input-output labels, and new labels are written in accordance with the conventions for input-output label writing.
9. When a file is opened in EXTEND mode
  - The current volume pointer is set to indicate the unit containing the last data record.

- If the file is an optional file and is not present, it is created. The current volume pointer is set to indicate the new unit.
- The file position indicator is positioned immediately after the last data record in the file; for an Indexed file, after the record with the highest key value.
- If label records are present and the file is on a single unit, the beginning file labels are processed.
- If label records are present and the file is not on a single unit, the beginning labels on the last unit are processed as though the file were being opened in INPUT mode. The existing ending file labels are processed as though the file were being opened in the INPUT mode. These labels are then deleted and processing proceeds as though the file had been opened in OUTPUT mode.

10. When a file is opened in SHARED mode

- If the file is a Sequential file, it must have variable length records. The only valid INPUT-OUTPUT statement for it is WRITE. If the file does not exist, it is created as a log file.
- If the file is Relative or Indexed, multiple users can modify records concurrently.
- For a Relative file, the file position indicator is set to 0. For an Indexed file, the file position indicator is set to the characters with the lowest position in the collating sequence associated with the file, and the prime record key is established as the key of reference.

11. When a file is opened in SPECIAL-INPUT mode

- Records are read without going through the sharer. While this yields much faster read times, the data obtained may not reflect the actual contents of the file.
- READ statements must use the prime record key.

12. A file is available if it is on a storage medium that is available to the machine and can be located by the data management system.

The following table shows the results of opening available and unavailable files.

**Availability of an Indexed File**

Open Mode	File Is Available	File Is Unavailable
INPUT or SPECIAL-INPUT	Normal OPEN	OPEN is unsuccessful
INPUT or SPECIAL-INPUT (optional file)	Normal OPEN	Normal OPEN, the first READ causes the AT END or invalid key condition
I-O or SHARED	Normal OPEN <sup>a</sup>	OPEN is unsuccessful <sup>a</sup>
I-O or SHARED (optional file)	Normal OPEN <sup>a</sup>	OPEN causes the file to be created <sup>a</sup>
OUTPUT	Normal OPEN; the file contains no records	OPEN causes the file to be created <sup>a</sup>
EXTEND	Normal OPEN	OPEN is unsuccessful
EXTEND (optional file)	Normal OPEN	OPEN causes the file to be created
<sup>a</sup> See the preceding General Rule 10 for Sequential files.		

13. The minimum and maximum record sizes for a file are established at the time the file is created (that is, when opened in OUTPUT mode) and cannot subsequently be changed.
14. If more than one file-name is specified in an OPEN statement, the file-names are opened in the left-to-right order in which they appear in the statement.
15. If a file attribute conflict condition exists, the OPEN statement is unsuccessful.
16. The OPEN statement updates the file status data item.
17. The NO REWIND phrase can be used only for sequential tape files on a single reel or, in a multiple file tape environment, for files wholly contained in a single reel of tape. The NO REWIND phrase is ignored if specified for a nontape file.
18. If the file is a tape file, and neither the EXTEND nor the NO REWIND phrase is specified, the OPEN statement positions the file at its beginning. If the NO REWIND phrase is specified, the file is not positioned; it must already be positioned at its beginning prior to execution of the OPEN statement.
19. A file on a multiple-file tape is treated the same as a file on a single-file tape. However, if multiple files are on a single reel of tape, only one of them can be open at any given time. There is, however, no constraint on the order in which files can be opened in INPUT mode.

## PERFORM Statement

The PERFORM statement transfers control to a procedure and gets control back when the procedure has been completed. It also can execute one or more statements written within the PERFORM statement itself.

### Format 1

```
PERFORM [procedure-name-1 [THROUGH procedure-name-2] ]  
        [imperative-statement-1   END-PERFORM ]
```

### Format 2

```
PERFORM    [procedure-name-1 [THROUGH procedure-name-2] ]  
            { identifier-1 }  
            { integer-1   } TIMES  
  
            [ imperative-statement-1 END-PERFORM ]
```

### Format 3

```
PERFORM [ procedure-name-1 [ THROUGH procedure-name-2 ] ]  
        [ WITH TEST { BEFORE } ] UNTIL condition-1  
        [ AFTER ]  
        [ imperative-statement-1 END-PERFORM ]
```

### Format 4

```
PERFORM [procedure-name-1 [THROUGH procedure-name-2] ]  
        [ WITH TEST { BEFORE } ]  
        [ AFTER ]  
  
        VARYING { identifier-2 } FROM { identifier-3 }  
                { index-name-1 }      { literal-1 }  
                                     { index-name-2 }  
  
        BY { identifier-4 } UNTIL condition-1  
           { literal-2   }  
  
        [ AFTER { identifier-5 } FROM { identifier-6 }  
          { index-name-3 }      { literal-3 }  
                              { index-name-4 }  
  
        BY { identifier-7 } UNTIL condition-2 ]...  
           { literal-4   }  
  
        [ imperative-statement-1   END-PERFORM ]
```



### *Syntax Rules*

1. PERFORM statements with procedure-name-1 specified are out-of-line PERFORM statements. PERFORM statements without procedure-name-1 specified are inline PERFORM statements.
2. When procedure-name-1 is omitted, imperative-statement-1 and the END-PERFORM phrase must be specified; when procedure-name-1 is specified, imperative-statement-1 and the END-PERFORM phrase cannot be specified.
3. When neither TEST BEFORE nor TEST AFTER is specified, TEST BEFORE is assumed.
4. Each data item must be numeric; identifier-4 and identifier-7 cannot have a value of zero. In format 2, identifier-1 must be a numeric integer.
5. Each literal must be numeric; a literal in a BY phrase cannot be zero.
6. When an index-name is specified in the VARYING or AFTER phrase
  - A data item in an associated FROM or BY phrase must be an integer.
  - A literal in an associated FROM phrase must be a positive integer.
  - A literal in an associated BY phrase must be a nonzero integer.
7. When an index-name is specified in the FROM phrase
  - A data item in an associated BY, VARYING, or AFTER phrase must be an integer.
  - A literal in the associated BY phrase must be an integer.
8. If either procedure-1 or procedure-2 is a declarative, the other must be in the same declarative section.
9. In format 4, up to six AFTER phrases can be specified.
10. In format 4, condition-1 and condition-2 can be any conditional expression; if procedure-name-1 is not used, the AFTER phrase cannot be used.
11. The END-PERFORM phrase delimits the scope of an inline PERFORM statement.

### *General Rules*

1. Unless stated to the contrary, the general rules apply to both inline and outof-line PERFORM statements.
2. The statements in the range of procedure-name-1 through procedurename-2, for an out-of-line PERFORM statement, or those contained in the PERFORM statement itself, for an inline PERFORM statement, are referred to as the specified set of statements.
3. Execution of the PERFORM statement transfers control to the first statement of the specified set of statements. The number of times that this transfer of control occurs depends on the following types of PERFORM statements:

- In a format 1 PERFORM, the specified set of statements is executed once, and then control passes to the end of the PERFORM statement.
  - In a format 2 PERFORM, the specified set of statements is executed the number of times specified by integer-1 or by the value of identifier-1 when execution of the statement commences. Following the execution of these statements for the specified number of times, control is transferred to the end of the PERFORM statement. If, when execution of the PERFORM statement begins, the value of identifier-1 is zero or negative, control passes to the end of the PERFORM statement. The number of executions of the specified set of statements is not altered if identifier-1 is changed during execution of the PERFORM statement.
  - In a format 3 PERFORM, the specified set of statements is performed until condition-1 is true. When the condition is true, control is transferred to the end of the PERFORM statement. Any subscripting or reference modification associated with the operands specified in condition-1 is evaluated each time the condition is tested. If condition-1 is true when the PERFORM statement is entered, and TEST BEFORE is in force, control immediately passes to the end of the PERFORM statement. If TEST AFTER is in force, the specified set of statements is executed once, and then control passes to the end of the PERFORM statement.
  - In a format 4 PERFORM, the specified set of statements is performed until one or more conditions is true.
4. The range of a PERFORM statement consists not only of the specified set of statements but also any statements executed as a result of the transfer of control by CALL, EXIT, GO TO, and PERFORM statements contained in the specified set of statements. The preceding is also true for all statements in declarative procedures that are executed as a result of the execution of statements in the range of the PERFORM statement. The statements in the range of a PERFORM statement need not appear consecutively in the program.
  5. When the range of a PERFORM statement includes another PERFORM statement, the range of the included PERFORM must either be totally included in, or totally excluded from, the range of the first PERFORM. Thus, an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement, cannot allow control to pass to the exit of the other active PERFORM. Furthermore, two or more such active PERFORM statements cannot have a common exit.
  6. If an EXIT PROGRAM statement is specified in the same program as the PERFORM, and the EXIT PROGRAM statement is within the range of the PERFORM statement, statements executed as the result of the transfer of control caused by the EXIT PROGRAM statement are not considered to be part of the range of the PERFORM.
  7. If a paragraph or section specified in a PERFORM statement has control passed to it by means other than a PERFORM statement (as, for instance, by the normal, sequential execution of the program), control passes through the last statement to the next executable statement.

8. In a format 4 PERFORM, the values of one or more data items or indexnames are varied during execution of the PERFORM statement. If either index-name-1 or index-name-3 is specified, the value of the associated index at the beginning of the PERFORM statement must be a valid occurrence number. If index-name-2 or index-name-4 is specified, the value of identifier2 or identifier-5 at the beginning of the PERFORM statement must be a valid occurrence number of an element in a table associated with index-name-2 or index-name-4. The following rules apply:
  - Subsequent augmentation of index-1 or index-3 must not result in the index being set to an invalid occurrence number. However, at the completion of the PERFORM statement, index-1 might contain a value that is one more than the greatest valid occurrence number or one less than the smallest valid occurrence number.
  - When identifier-2 or identifier-5 is subscripted, the subscripts are evaluated each time the associated data item is set to a value or augmented. If identifier-3, identifier-4, identifier-6, or identifier-7 is subscripted, the subscripts are evaluated each time the associated data item is used in a setting or augmenting operation.
  - Any subscripting or reference modification associated with the operands specified in condition-1 or condition-2 is evaluated each time the condition is tested.
9. When the specified set of statements has been executed, an implicit transfer of control is made back to the end of the PERFORM statement. The following rules apply:
  - If procedure-name-2 is not specified and procedure-name-1 is a paragraph-name, this transfer occurs after the last statement of procedure-1; if procedure-name-1 is a section-name, the transfer occurs after the last statement of the last paragraph in procedure-1.
  - If procedure-name-2 is specified and it is a paragraph-name, this transfer occurs after the last statement of the paragraph. If it is a section-name, the transfer is after the last statement of the last paragraph in the section.
  - For an inline PERFORM statement, execution is completed after the last statement contained in the PERFORM has been executed.
10. There need be no relationship between procedure-1 and procedure-2, except that a consecutive sequence of operations executed beginning at procedure-1 eventually appears in procedure-2. In particular, GO TO and PERFORM statements can occur between the beginning of procedure-1 and the end of procedure-2. If there are two or more logical paths to the return point, procedure-name-2 can be the name of a paragraph consisting of the EXIT statement to which all of these paths lead.
11. Procedure-name-1 and procedure-name-2 cannot be the names of sections or paragraphs in another program in the run unit, irrespective of whether or not the other program contains or is contained within the program containing the PERFORM statement. Statements in other programs in the run unit can be executed as a result of executing a PERFORM statement only if the range of that PERFORM statement includes a CALL statement.

## READ Statement

The READ statement makes a record of a file available to the program.

### Format

```
READ file-name-1 [NEXT] RECORD [ MODIFIABLE  
                                WITH HOLD  
                                ALTERED ]  
  [ INTO data-name-1 ]  
  [ KEY IS data-name-2 ]  
  
  [ TIMEOUT OF { data-name-3 } [ SECOND  
                                integer-1 ] [ SECONDS ]  
  
    [ HOLDER-ID IN data-name-4 ]  
  
    { imperative-statement-1 } ]  
  [ NEXT SENTENCE ]  
  
  [ { INVALID KEY } imperative-statement-2 ]  
    [ AT END ]  
  
  [ NOT { INVALID KEY } imperative-statement-3 ]  
    [ AT END ]  
  
[END-READ]
```

### Syntax Rules

1. The record area of file-name- I cannot overlap that of data-name-1.
2. The NEXT phrase must be specified for files in DYNAMIC ACCESS mode when records are retrieved sequentially. If the ACCESS mode is sequential, this phrase is optional.
3. The ALTERED and MODIFIABLE phrases can be specified only with files whose device is display.
4. If file-name- I identifies an indexed file, data-name-2 must be a record key associated with file-name-1; it can be qualified.
5. The INTO phrase can be specified only if the file has only one record description, or, in the case of more than one record description, if all record-names, as well as data-name-1, are either group items or elementary alphanumeric items.
6. The KEY phrase cannot be specified for files with SEQUENTIAL ACCESS mode, or when the NEXT phrase is specified
7. The TIMEOUT phrase can be specified only if the organization of file-name1 is indexed and if the HOLD phrase is specified.
8. Data-name-3 must be an integer data item. Integer-1 must have a value from 1 to 255.

9. Data-name-4 must be defined in the Working-Storage section or Linkage section and have a PICTURE of X(3).
10. The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for file-name-1.
11. If file-name-1 identifies a DISPLAY device, the AT END phrase can be specified only when the ALTERED phrase is specified.

### ***General Rules***

1. File-name-1 must be open in INPUT, I-O, SHARED, or SPECIAL-INPUT mode.
2. If the READ is successful, that is, if no exception condition exists, the record is put into the record area of the file and, if INTO is specified, also into dataname-1. Then, if imperative-statement-2 is not specified, control passes to the end of the READ statement. If imperative-statement-2 is specified, execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the READ statement. (The record is available to the program prior to the execution of imperative-statement-2 or any statement following the READ statement.)
3. If the number of characters in the record is less than the minimum size specified for the file, the portion of the record area that is to the right of the last character read is undefined; in particular, it cannot be assumed to contain spaces. If the number of characters in the record is greater than the maximum size specified for the file, the record is truncated on the right. In either of these cases, the READ statement is successful, but the file status data item is set to indicate that a record length conflict has occurred.
4. If the file has more than one record description, the various record types share the same record area in storage. (This is an implicit redefinition of the record area.) Any data in this area that is beyond the limits of the current data record is undefined.
5. If the file position indicator indicates that an optional input file is not present or no next record exists, the AT END condition exists, and the file status data item is updated to indicate this condition under the situations in General Rules 6 and 7.
6. If an exception condition occurs, the READ statement is unsuccessful. The file status data item is updated, and execution continues according to the rules for exception handling.
7. If the READ statement is unsuccessful, the record area of the file does not have valid data, and the file position indicator is set to indicate that no valid next record has been established.
8. A READ statement with the INTO phrase is the same as the READ statement without the INTO phrase, followed by a move of the record just read from the record area to data-name-1, according to the rules for the MOVE statement without the CORRESPONDING phrase. (This implicit move does not occur if execution of the

READ statement is unsuccessful.) The size of the record is determined by the file's RECORD clause. ,

Any subscripting associated with data-name-1 is evaluated after the record has been read but before it is moved to the data item.

9. The HOLD phrase can be used only with a file opened in SHARED or I-O mode. Following successful execution of a READ WITH HOLD statement, the record read can be accessed by other users sharing that file, but it cannot be modified by them until it is released. The following rules apply:
  - If another record (in the same or another file) is being held by the program, that record is released upon execution of the READ WITH HOLD for a new record.
  - If the record is being held by another task, the **program** must wait until that record is released.
  - A program cannot hold one file and issue a READ WITH HOLD for a record in another file.
  - A held record is released by execution of any DELETE, REWRITE, or WRITE statement; by execution of a READ WITH HOLD statement for the same or another file; or by execution of a CLOSE statement for the file. (This includes the implicit CLOSE executed by the STOP RUN statement.)
10. If the TIMEOUT phrase is specified and the read operation cannot be completed in the amount of time specified by data-name-1 or integer-1, imperative-statement-1 is executed. If a timeout occurs, and the HOLDER-ID phrase is specified, the logon initials of the user currently holding the desired resources are moved to the data-name-2.
11. If the Compiler option COMPAT14 is YES, and the file is opened in SHARED or I-O mode with an ACCESS mode of DYNAMIC, a READ WITH HOLD statement must be executed before a DELETE or REWRITE statement is executed for the file.
12. The END-READ phrase delimits the scope of the READ statement.
13. For an Indexed file being accessed sequentially in SEQUENTIAL or DYNAMIC ACCESS mode, the file position indicator determines the record to be read. The following rules apply:
  - If the file position indicator was established by a previous OPEN or START statement, the first existing record in the file whose current key of reference value is greater than or equal to the file position indicator is selected.
  - If the file position indicator was established by a previous READ statement, and the current key of reference does not allow duplicates, the first existing record in the file whose corresponding key value is greater than the file position indicator is selected. If the current key of reference does allow duplicates, the first record in the file

whose key value is either equal to the file position indicator and whose position within the set of duplicates is immediately after the record that was read by that previous READ statement is selected. If no such record exists, the first record whose key value is greater than the file position indicator is selected.

14. If the READ of an Indexed file is successful, the file position indicator is set to the value and reference of the key whose record was read.
15. For a sequential or relative file accessed sequentially in SEQUENTIAL or DYNAMIC ACCESS mode, the file position indicator determines the record to be read. The first record in the file whose record number is greater than the file position indicator is read.
16. If the READ of a sequential or relative file is successful, the file position indicator is set to the record number of the record that was read. If filename-1 has a RELATIVE KEY data item associated with it in its FILECONTROL entry, the relative number of the record just read is placed in it.
17. If the KEY phrase is not specified for an Indexed file being accessed randomly in RANDOM or DYNAMIC ACCESS mode, the prime record key is established as the key of reference. If the KEY phrase is specified, dataname-2 is established as the key of reference. The file position indicator is set to the value of the key of reference. This value is then compared with the key values of the records in the file until a record having an equal value is found.
18. When records are being accessed randomly for a relative or sequential file in RANDOM or DYNAMIC ACCESS mode, the record in the file whose relative record number equals the value of the file's relative key data item is read. If a record with this number does not exist in the file, the INVALID KEY condition exists and the READ statement is unsuccessful.
19. With hashed Extended DMS (XDMS) files on the VS, executing a READ NEXT statement gives unpredictable results.
20. If the end of a unit is recognized, or a unit contains no logical records and the logical end of the file has not been reached, the standard ending unit label procedures are performed. Next, a unit swap occurs, the current volume pointer is updated to point to the next unit in the file, and the standard beginning unit label procedure is performed on this new unit.
21. If an AT END condition occurs, another READ statement cannot be executed for file-name- 1 without resetting the file position indicator.
22. A record of a workstation file consists of a 4-byte *order area* followed by a *mapping area* of up to 1,920 data bytes.
23. Upon completion of a READ of a workstation file, the record is available in the file's record area. The current cursor position is in both the cursor position data item and bytes 2 and 3 of the order area, and the Attention Identifier (AID) characters are in

the second byte of the file status data item. Refer to Appendix E to see a table of AID characters.

24. If a nonexistent row is specified, execution continues with imperative-statement-2 and according to the rules for exception handling.
25. If MODIFIABLE is specified, the contents of the modifiable locations of the workstation screen are moved to the mapping area. Pseudo-blank characters involved in the READ are changed to blanks before being moved, and any blinking characters that are read are changed to high-intensity nonblinking characters both on the screen and in the mapping area.
26. If ALTERED is specified, only those fields that were changed are moved to the mapping area. If the AT END phrase is specified and no field has been changed, imperative-statement-1 is executed, and execution continues according to the rules for exception handling.
27. If the Compiler option COMPAT74 is NO and a workstation error occurs, the values in the cursor position data item and the PF key data item have no significance.



## RELEASE Statement

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

### *Format*

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

### *Syntax Rules*

1. Record-name-1 must be in a sort-merge file; it can be qualified.
2. A RELEASE statement can be used only in an input procedure of a SORT statement for the file that contains record-name-1.
3. The storage area of record-name-1 cannot overlap with that of data-name-1.

### *General Rules*

1. The RELEASE statement passes a record to the initial phase of a sort operation. This record is usually no longer available in the file's record area. However, if the file is specified in a SAME RECORD AREA clause, the record is available both in the file's record area and also as a record of any other file specified in that clause that is currently open in the OUTPUT mode.
2. When the FROM phrase is specified, data-name-1 is implicitly moved to record-name-1, and then the record is passed to the input routine. At the end of execution of the RELEASE statement, the data is still available in dataname-1, even though it may not be available in the file's record area.

## RETURN Statement

The RETURN statement obtains sorted records from the final phase of a sort operation or merged records from a merge operation.

Format

```
RETURN file-name-1 RECORD [ INTO data-name-1 ]  
  
    AT END imperative-statement-1  
  
    [ NOT AT END imperative-statement-2 ]  
  
    [ END-RETURN ]
```

### *Syntax Rules*

1. File-name-1 must identify a sort-merge file.
2. The storage area of data-name-1 cannot overlap the record area of the file.
3. A RETURN statement can be used only in the range of an output procedure associated with a SORT or MERGE statement for file-name-1.
4. The INTO phrase can be specified *only* if the file has only one record description, or, in the case of more than one record description, if all recordnames, as well as data-name-1, are either a group item or an elementary alphanumeric item.

### *General Rules*

1. If the file has more than one record type, its records share the same storage area. This is equivalent to an implicit redefinition of the area. Any data items beyond the range of the current data record are undefined at the completion of the RETURN statement.
2. After the sorting or merging phase has been completed, the RETURN statement puts the next record in the file-name-1 record area in order by the key or keys specified in the SORT or MERGE statement. If no "next" record exists, the AT END condition occurs, the RETURN is unsuccessful, and control passes to imperative-statement-1.
3. When the AT END condition exists, the file-name-1 record area does not contain valid data, and execution continues with each statement in imperative-statement-1. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-1, control is transferred to the end of the RETURN statement.
4. When the AT END condition does not occur, the record is available in the record area of the file or to the end of the RETURN statement.

5. If the RETURN statement has the INTO phrase, the record just read is moved from the record area to data-name-1, according to the rules for the MOVE statement. This implicit move does not occur if execution of the RETURN statement is unsuccessful.

Any subscripting necessary for data-name-1 is evaluated after the record has been read and before it is moved to data-name-1.

6. The END-RETURN phrase delimits the scope of the RETURN statement.

## REWRITE Statement for Sequential Files

The REWRITE statement replaces an existing record.

### **Format**

```
REWRITE record-name-1 [FROM data-name-1] [AFTER  
{  
  ALARM  
  ROLL DOWN  
  ROLL UP  
  ERASE PROTECT  
  ERASE MODIFY  
  SETTING CURSOR COLUMN {data-name-2} {ROW} {data-name-3}  
                        {integer-1} {LINE} {integer-2}  
}]  
[ END-REWRITE ]
```

### *Syntax Rules*

1. The record-name-1 storage area cannot overlap data-name-1.
2. Record-name-1 can be qualified.
3. The AFTER clause is valid only when the file associated with record-name-1 is a workstation file.
4. Data-name-2 must identify an integer data item. Integer-1 can range from 1 through 80.
5. Data-name-3 must identify an integer data item. Integer-2 can range from 1 through 24.

### *General Rules*

1. File-name-1 must be either a mass storage or workstation file open in I-O mode. It cannot be compressed.
2. If the statement has the FROM phrase, the record is transferred from data-name-1 to the file's record area, according to the rules for the MOVE statement, and then rewritten. After the REWRITE has been completed, the record is available in data-name-1, even if it is not available in the file's record area. Any subscripting associated with data-name-1 is evaluated before the data is moved to the record area.
3. The number of character positions in record-name-1 must be equal to the number of character positions in the record being replaced. If they are not equal, the REWRITE statement is unsuccessful, the updating operation does not take place, the content of the file's record area is unaffected, and the file status data item is set to a value indicating this situation.
4. The REWRITE statement does not affect the file position indicator.
5. The END-REWRITE phrase delimits the scope of the REWRITE statement.
6. The last INPUT-OUTPUT statement executed for the file associated with record-name-1 must have been a successful READ statement. The record that was read is the one that is replaced.
7. If the REWRITE is successful, that is, if no exception condition occurs, the rewritten record is no longer available in the file's record area. However, if the file is specified

in a SAME RECORD AREA clause, the record is available both in the file's record area and also as a record of any other file specified in the clause that is currently open in the OUTPUT mode.

8. A workstation REWRITE statement moves the relative record number to the first byte of the order area and then writes data to the workstation screen as determined by the AFTER phrase.
9. The function specified in the AFTER phrase determines the Write Control Character (WCC) used in the REWRITE operation. For additional details on the WCC, refer to Appendix D.
10. If ROLL DOWN or ROLL UP is specified, record-name-1 must be only one line.
11. If ROW (or LINE) is not specified, the line that is rewritten is specified by the record's relative key.
12. If the Compiler option COMPA774 is NO and a workstation error occurs, the values in the cursor position data item and the PF key data item are meaningless.

## REWRITE Statement for Relative and Indexed Files

The REWRITE statement replaces an existing record.

Format

```
REWRITE record-name-1 [ FROM data-name-1 ]  
    [ INVALID KEY imperative-statement-1 ]  
    [ NOT INVALID KEY imperative-statement-2  
[ END-REWRITE
```

### *Syntax Rules*

1. The record-name-1 storage area cannot overlap data-name- 1.
2. Record-name-1 can be qualified
3. The INVALID KEY and the NOT INVALID KEY phrases cannot be specified if the file is in SEQUENTIAL ACCESS mode.
4. The INVALID KEY phrase must be specified if the file is in the RANDOM or DYNAMIC ACCESS mode unless there is an applicable USE procedure, in which case the phrase is optional.

### *General Rules*

1. The file associated with record-name-1 must be open in I-O or SHARED mode.
2. If the file has SEQUENTIAL access, the last I-O statement executed for it must have been a successful READ statement. The record that was read is the one that is replaced. If the file has indexed organization at the time of the rewrite operation, the prime record key must contain the key value for this record.
3. If the file has RANDOM or DYNAMIC access, the record replaced is specified by the contents of the prime record key for indexed files or the contents of the relative key data items for relative files. If the file does not contain this record, the INVALID KEY condition exists, the updating operation does not take place, the contents of the record area are unaffected, and the file status data item is set to indicate this condition.
4. For an indexed file open for SEQUENTIAL access in the SHARED mode, if the COMPA774 Compiler option is set to YES or I-O, the last INPUT/OUTPUT statement to affect the file prior to the execution of the REWRITE statement must be a successful READ WITH HOLD. If the COMPAT74 option is set to NO, the same rule applies with the exception that the WITH HOLD clause is not required.
5. If an indexed file record has alternate keys, the order of retrieval remains unchanged, unless the value of an alternate key of reference is changed.
6. If the REWRITE is successful, that is, if no exception condition exists, the rewritten record is no longer available in the record area of the file. However, if the file is specified in a SAME RECORD AREA clause, the record is available both in the

area of the file and also as a record of any other file specified in the clause that is currently open in the OUTPUT mode.

If imperative-statement-2 is not specified, control passes to the end of the REWRITE statement. If imperative-statement-2 is specified, execution continues with each statement in it. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the REWRITE statement.

7. An INVALID KEY condition occurs under the following conditions:
  - The ACCESS mode is RANDOM or DYNAMIC, and the value of the prime record key for an indexed file, or the relative key, data item for a relative file is not equal to the value of any record in the file.
  - The ACCESS mode is sequential, and the value of the prime record key is not equal to the value of the last record read.
  - The value of an alternate key for which duplicates are not allowed is equal to the value of that key for another record in the file.
8. If the INVALID KEY condition occurs, execution of the REWRITE statement is unsuccessful. If the INVALID KEY phrase is specified, execution continues with each statement in imperative-statement-1. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-1, control is transferred to the end of the REWRITE statement. Any associated USE AFTER EXCEPTION procedure is not executed. If the INVALID KEY phrase is not specified, the applicable USE procedure is executed, and control returns to the next executable statement following the REWRITE.
9. If an exception condition other than an INVALID KEY condition occurs, and there is an applicable USE statement, it is executed, and upon its completion control returns to the statement following the REWRITE. If a USE procedure is not specified, the program crashes.
10. If the REWRITE statement is unsuccessful, the record area of the file is not affected.
11. If the statement has the FROM phrase, the record is transferred from dataname-1 to the record area of the file, according to the rules for the MOVE statement, and then rewritten. After the REWRITE has completed, the record is available in data-name-1, even if it is not available in the record area of the file. Any subscripting associated with data-name-1 is evaluated before the data is moved to the record area.
12. The REWRITE statement updates the FILE STATUS data item. For an indexed file, it does not affect the file position indicator.
13. For an indexed file, the number of character positions in record-name-1 must be equal to the number of character positions in the record being replaced. If they are not equal, the REWRITE statement is unsuccessful, the updating operation does not take place, the content of the record area of the file is unaffected, and the file status data item is set to a value indicating this situation. However, if the record is a compressed record, the rewritten record can have a different length.

14. If the Compiler option COMPAT74 is NO and integer-1 is not specified in the ALTERNATE KEY clause for the file, the character positions designated by the data-name specified in the ALTERNATE KEY clause are the same for all record types in that file.
15. If the Compiler option COMPAT74 is I-O or YES or if integer-1 is specified in the ALTERNATE KEY clause for the file, the alternate index paths through which the rewritten record is able to be accessed are indicated by the record keys associated with record-name-1. When rewritten, the record is deleted from any current paths not indicated by an ALTERNATE KEY clause in the FILE-CONTROL entry of the file and is added to any necessary new paths.
16. The END-REWRITE phrase delimits the scope of the REWRITE statement.



## ROLLBACK Statement

The ROLLBACK statement removes all updates and releases all locks acquired since the corresponding BEGIN statement.

### Format

```
ROLLBACK [ data-name-1 ]  
           [ literal-1 ]  
  
           [ ON ERROR      imperative-statement-1 ]  
           [ NOT ON ERROR imperative-statement-2 ]  
  
           [END-ROLLBACK]
```

### Syntax Rule

Data-name-1 must be a numeric integer. Literal- I must be a positive integer.

### General Rules

1. Rollback of a top-level transaction causes all locks in effect for that transaction to be released.
- Literal-1 or data-name- I indicates the number of transaction levels to roll back. When neither is specified, all levels are rolled back.
- If the ROLLBACK statement is successful and the NOT ERROR phrase is specified, execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the ROLLBACK statement.
- If the ROLLBACK statement is unsuccessful, execution continues according to the rules for exception handling.
- The special register, RETURN-CODE, can contain the following values:

Code	Meaning
0	Success
4	Files not transaction capable
8	Transactions not supported on this system
20	I-O error while accessing Before Image Journal
24	I-O error while accessing file data
32	Unable to set file crash status

## SEARCH Statement

The SEARCH statement searches a table for an element that satisfies a specified condition and adjusts the value of an index to indicate that table element.

### Format 1

```
SEARCH data-name-1 [ VARYING { data-name-2 }  
                      { index-name-1 } ]  
[AT END imperative-statement-1]  
{ WHEN condition-1 { imperative-statement-2 } } ...  
  { NEXT SENTENCE }  
[END-SEARCH]
```

### Format 2

```
SEARCH ALL data-name-1 [AT END imperative-statement-1]  
WHEN { data-name-2 { IS EQUAL TO } { data-name-3 }  
        { IS = } { literal-1 } } ...  
        { condition-name-1 }  
[ { AND { data-name-4 { IS EQUAL TO } { data-name-5 }  
          { IS = } { literal-2 } } } ...  
  { condition-name-2 } ]  
{ imperative-statement-2 }  
{ NEXT SENTENCE }  
[ END-SEARCH ]
```

### Syntax Rules

1. Data-name-1 cannot be subscripted or indexed
2. The description of data-name-1 must contain an OCCURS clause with the INDEXED phrase, and, for a format 2 SEARCH statement, the KEY phrase.
3. If a data-name in the KEY phrase in the data-name OCCURS clause, or a condition-name associated with such a data-name, is referenced, all preceding data-names in that KEY phrase or their associated conditionnames must also be referenced.
4. If the END-SEARCH phrase is specified, the NEXT SENTENCE phrase cannot be specified

### Format 1

5. Data-name-2 must be an index data item or a numeric integer. It cannot be subscripted by the first index-name specified in the INDEXED BY phrase in the data-name-1 OCCURS clause.

6. Condition-1 can be any conditional expression.

#### *Format 2*

7. All condition-names must have a single value. The data-name associated with a condition-name must appear in the KEY phrase in the data-name-1 OCCURS clause.
8. Data-name-2 and data-name-4 can be qualified and must be subscripted by the first index-name associated with data-name-1 (along with other subscripts as required) and must be referenced in the KEY phrase in the data-name-1 OCCURS clause.
9. Data-name-3, data-name-4, or the data-names specified in an arithmetic expression cannot be specified in the KEY phrase in the data-name-1 OCCURS clause. In addition, they cannot be indexed by the first index-name associated with data-name-1.

#### *General Rules*

1. The scope of a SEARCH statement is terminated by an END-SEARCH phrase at the same level of nesting, by a separator period, or by the termination of an enclosing imperative statement.

#### *Format 1*

2. A format 1 SEARCH is a serial search. A serial search starts with the current index setting and proceeds as follows:

If, at the start of execution of the SEARCH statement, the index name associated with data-name-1 contains a value that is greater than the highest valid occurrence number for that data-name, the SEARCH terminates, and if the AT END phrase is not specified, control passes to the end of the SEARCH statement. If the AT END phrase is specified, execution continues with each statement in imperative-statement-1. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-1, control is transferred to the end of the SEARCH statement.

Otherwise, the conditions are evaluated in the order in which they are written. This evaluation makes use of index settings, if specified, to determine which instance of those items is tested. If none of the conditions is satisfied, the index-name for data-name-1 is incremented to indicate the next higher occurrence. If this new value is not valid, the search terminates and action is as described previously. If it is valid, the above process is then repeated, using the new index settings.

When any one of the conditions is satisfied, the search terminates, and the imperative statement associated with that condition is executed. If one of its statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of the imperative-statement, control is transferred to the end of the SEARCH statement.

If, instead of an imperative-statement, the NEXT SENTENCE phrase is specified, control passes to the sentence following the SEARCH statement. In either case, the index-name remains set at the occurrence that caused the condition to be satisfied.

3. If the VARYING phrase is not specified, the index name used for the search operation is the first index name in the INDEXED BY phrase of dataname-1. Any other index names for data-name-1 remain unchanged.
4. If the VARYING phrase is specified with an index name, and if the index name is in the INDEXED BY phrase in data-name-1 OCCURS clause, that index name is used for this search. If not, or if the VARYING phrase is specified with a data-name, the first index name in the INDEXED BY phrase in the data-name-1 OCCURS clause is used for the search.

If the VARYING phrase specifies an index name and that name is in the INDEXED BY phrase of a different data-name, the occurrence number represented by that index is incremented by the same amount as the index associated with data-name-1.

If the VARYING phrase specifies a data-name, and if the data-name is an index data item, it is incremented by the same amount as the index associated with data-name-1. If the data-name is not an index data item, it is incremented by the value 1.

#### Format 2

5. Format 2 is a binary search. The initial value of the index associated with data-name-1 is ignored. During the search, its setting is varied such that at no time is it set to a value that is less than 1 or greater than the highest valid occurrence number for the table.
6. If none of the specified conditions specified in the WHEN clause can be satisfied for any setting of the index, and the AT END phrase is specified, execution continues with each statement in imperative-statement-1. If one of these statements causes an explicit transfer of control, control is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-1, control is transferred to the end of the SEARCH statement. If the AT END phrase is not specified, control passes to the end of the SEARCH statement. In either case, the final setting of the index is not predictable.
7. If all the conditions can be satisfied, the index indicates a table element that satisfied them, and, if NEXT SENTENCE is specified, control passes to the sentence following the SEARCH statement. If imperative-statement-2 is specified, execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the SEARCH statement.

8. The results of the search operation are predictable only when the data in the table is ordered in the same manner as specified in the `KEY IS` phrase of the `data-name-1 OCCURS` clause, and the key(s) specified in the `WHEN` phrase can identify a unique table element.
9. The index used for the search operation is the first one specified in the `INDEXED BY` phrase in the `data-name-1 OCCURS` clause. Any other indexes associated with `data-name-1` are not affected by this operation.

## SET Statement

The SET statement assigns or modifies an index data item or index, assigns a status to a mnemonic name, sets a condition name true, or sets a figurative constant to a value.

### Format 1

$$\underline{\text{SET}} \left\{ \begin{array}{l} \text{index-name-1} \\ \text{data-name-1} \end{array} \right\} \dots \underline{\text{TO}} \left\{ \begin{array}{l} \text{index-name-2} \\ \text{data-name-2} \\ \text{integer-1} \end{array} \right\}$$

### Format 2

$$\underline{\text{SET}} \{ \text{index-name-3} \dots \left\{ \begin{array}{l} \underline{\text{UP BY}} \\ \underline{\text{DOWN BY}} \end{array} \right\} \left\{ \begin{array}{l} \text{data-name-3} \\ \text{integer-2} \end{array} \right\}$$

### Format 3

$$\underline{\text{SET}} \{ \text{mnemonic-name-1} \} \dots \underline{\text{TO}} \left\{ \begin{array}{l} \underline{\text{ON}} \\ \underline{\text{OFF}} \end{array} \right\}$$

### Format 4

$$\underline{\text{SET}} \{ \text{condition-name-1} \} \dots \underline{\text{TO TRUE}}$$

### Format 5

$$\underline{\text{SET}} \text{ figurative-constant-1} \\ \left\{ \left\{ \begin{array}{l} \underline{\text{IN}} \\ \underline{\text{OF}} \end{array} \right\} \underline{\text{EAC OF}} \right\} \left\{ \begin{array}{l} \text{display-item-1} \\ \text{data-name-4} \end{array} \right\} \left\{ \begin{array}{l} \underline{\text{ON}} \\ \underline{\text{OFF}} \end{array} \right\}$$

### Format 6

$$\underline{\text{SET}} \left\{ \begin{array}{l} \underline{\text{ADDRESS OF based-data-name-1}} \\ \text{pointer-item-1} \end{array} \right\} \dots \underline{\text{TO}} \left\{ \begin{array}{l} \underline{\text{ZERO}} \\ \underline{\text{ADDRESS OF data-name-2}} \\ \text{pointer-item-2} \end{array} \right\}$$

### Syntax Rules

#### Formats 1 and 2

1. All references to index-name-1, data-name-1, and index-name-3 apply to all recursions of them.
2. Integer-1 and integer-2 can be signed. Integer-1 must be positive.

#### Format 1

3. Data-name-1 and data-name-2 must be an index data item or a numeric integer.

*Format 2*

4. Data-name-3 must be a numeric integer.

*Format 3*

5. Mnemonic-name-1 must be SWITCH-1 through SWITCH-7.

*Format 4*

6. Condition-name-1 must be associated with a conditional variable.

*Format 5*

7. Figurative-constant-1 must be a 1-byte data item defined in the FIGURATIVE-CONSTANTS paragraph.
8. Data-name-4 must be a 1-byte data item.
9. Display-item-1 must be a data item in a working storage record whose usage is DISPLAY-WS or be in a record of a file with a device type of "DISPLAY".

*Format 6*

10. Based-data-name-1 must be defined in the LINKAGE SECTION of the program and not be referenced in the Procedure Division header USING phrase.
11. Pointer-item-1 and pointer-item-2 must be defined as data names whose usage is POINTER.

## General Rules

### Format 1

1. If index-name-1 is specified, the SET statement must set index-1 to a valid occurrence number for its table. (In most cases, trying to set it to an invalid value sets it to 1, but not always.) Index-1 is set to the table element indicated by index-name-2, data-name-2, or integer-1. If index-name-2 is specified, index-2 must have a valid occurrence number for its table before execution of the SET statement. If index-name-2 is related to the same table as index-1, or if data-name-1 is an index data item, no conversion takes place, and the contents of index-2 or data-name-1 are moved directly to index-1. Consequently, it is possible for index-1 to have an invalid occurrence number. If the Compiler option SUBCHK is NO (refer to Appendix B), the program could access data outside the table.
2. If data-name-1 is an index data item, data-name-1 can be set to the contents of index-name-2 or to data-name-2. No conversion takes place in either case. If data-name-1 is not an index data item, it is set to the occurrence number that corresponds to the value of index-2. (Neither data-name-2 nor integer-1 can be used in this case.)
3. If more than one index-name-1 or data-name-1 is specified, the preceding process is repeated for each of them. The value of index-name-2 or data-name-2 is used as it was at the beginning of the execution of the statement, but any subscripting associated with data-name-1 is evaluated immediately before the value of data-name-1 is changed.
4. The following table shows the valid combinations of a format 1 SET statement:

SET Item Is	TO Item Is:			
	Index	Index Data Item	Integer Data Item	Integer
Index	Y	Y	Y	Y
Index Data Item	Y	Y	N	N
Integer Data Item	Y	N	N	N

### Format 2

5. Index-3 is incremented or decremented by a value that corresponds to the number of occurrences indicated by integer-2 or data-name-3. If there are multiple occurrences of index-3, the value that data-name-3 had at the beginning of the statement is used for each repetition. Index-3 must have a valid occurrence number for its table before and after execution of the SET statement.

### Format 3



6. If the external switch is put into the ON condition, a subsequent switchstatus condition for the switch is true. If the switch is put into the OFF condition, the test is false.

*Format 4*

7. The literal in the VALUE clause of condition-name-1 is put into the associated conditional variable. If more than one literal is specified in the VALUE clause, the conditional variable is set to the value of the first literal in the clause.
8. If multiple condition-names are specified, the results are the same as if a separate SET statement were written for each condition-name in the same order as they were specified in the SET statement.

*Format 5*

9. This format of the SET statement sets selected bits in a 1-byte data item. Bits in the specified item corresponding to bits that are a 1 in the figurative constant are set to 1 when ON is specified and to 0 when OFF is specified

*Format 6*

10. This format of the SET statement is used to provide an address to a based record or to set a pointer item to another pointer item or to zero.

## SORT Statement

The SORT statement creates a sort file either by executing an input procedure or by transferring records from another file. It then sorts the records by one or more keys and makes each record available, in sorted order, to an output procedure or file.

### Format

```
SORT file-name-1 { ON { ASCENDING  
                     DESCENDING } KEY {data-name-1}... } ...  
  
[ WITH DUPLICATES IN ORDER ]  
  
[ COLLATING SEQUENCE IS alphabet-name-1 ]  
  
{ INPUT PROCEDURE IS proc-name-1 [THROUGH proc-name-2] }  
{ USING {file-name-2}... }  
  
{ GIVING {file-name-3} ... }  
{ OUTPUT PROCEDURE IS proc-name-3 [THROUGH proc-name-4] }
```

### Syntax Rules

1. A SORT statement cannot appear in a Declarative section.
2. File-name-1 must be a sort-merge file.
3. If file-name-1 has variable-length records, the size of the records in file-name-2 cannot be smaller than the smallest record type nor larger than the largest record type in file-name-1. If file-name-1 has fixed-length records, the size of the records in file-name-2 cannot be larger than the largest record type in file-name-1.
4. Data-name-1 is the name of a key data item. Key data items are those on which the sort is based. Up to eight key data items can be specified; they can be qualified  
  
A key data item must be in a record of file-name-1. It cannot contain an OCCURS clause in its description or be subordinate to an item that contains an OCCURS clause. If it is a group item, it cannot contain a variable occurrence data item. If file-name-1 has more than one record description, key data items need be described in only one of the record descriptions. The same character positions are taken as the key in all the record types of the file.  
If file-name-1 contains variable-length records, all key data items must be contained in the first x character positions of the record, where x equals the size of the smallest record of file-name-1.
5. File-name-2 and file-name-3 cannot be sort-merge files.
6. File-name-2 and file-name-3 can be on the same multiple file reel.
7. If file-name-3 is an indexed file, the first data-name-1 must be ASCENDING and must occupy the same character positions in its record as the prime record key for file-name-3.

8. If file-name-3 contains variable-length records, the size of the records in filename-1 cannot be smaller than the smallest record or greater than the largest record size in file-name-3. If file-name-3 contains fixed-length records, the size of the records in file-name-1 cannot be larger than the largest record size in file-name-3.
9. No pair of file-names can be specified in a given SAME SORT AREA or SAME SORT-MERGE AREA clause. Files in the GIVING phrase cannot be specified in a given SAME clause.
10. A record cannot be greater than 2,024 bytes.

### ***General Rules***

1. The SORT statement has three phases:
  - Records are transferred to file-name-1 either by the execution of RELEASE statements in an input procedure or by the execution of implicit READ statements for file-name-2.
  - File-name-1 is put into a sorted order as a function of the keys specified.
  - The records of file-name-1, in sorted order, are either written to filename-3 or, upon execution of a RETURN statement, are made available for processing by an output procedure.
2. If file-name-1 contains only fixed-length records, any record in file-name-2 that contains fewer character positions is space filled on the right when that record is transferred to file-name-1.
3. The data-names following KEY are taken, from left to right, as having decreasing significance. The leftmost data-name is the major or most significant key; the next data-name is the next most significant key, and so on. This is without regard as to how they are divided among KEY phrases. In other words, the following phrases both specify KEY-1 as the most significant key, KEY-3 as the least significant key, and KEY-2 as the one in between:
 

ASCENDING KEY-1, KEY-2, KEY-3

ASCENDING KEY-1, ASCENDING KEY-2, DESCENDING KEY-3
4. If ASCENDING is specified, the sorted sequence is from the lowest values of the key data items to the highest, according to the rules for comparison of operands in a RELATION condition.
5. If DESCENDING is specified, the sorted sequence is from the highest values of the key data items to the lowest, according to the rules for comparison of operands in a RELATION condition.
6. If the DUPLICATES phrase is specified and the contents of all the key data items of one data record are equal to those of another, the sorted order is the same as the order in which the files are specified in the SORT statement. If the two records are in the same file, the sorted order is that in which the records are accessed in the file. If an input procedure is specified, the sorted order is that in which the records are released by the procedure. If the DUPLICATES phrase is not specified and the contents of all the key data items of one data record are equal to those of another, the sorted order is not predictable.

7. If the COLLATING SEQUENCE phrase is specified, that collating sequence is used in the comparison of nonnumeric key data items. If not, the program collating sequence is used.
8. If the USING phrase is specified all the records in file-name-2 are transferred to file-name-1. File-name-2 cannot be open when the SORT statement is executed. For each file in the phrase
  - An implicit OPEN INPUT statement is executed for the file.
  - The records of the file are released to the sort operation as if a READ NEXT statement with the AT END phrase were executed. If the file is a relative file, the content of the relative key data item is undefined after the execution of the SORT statement.
  - An implicit CLOSE statement without any optional phrases is executed for the file.
  - These implicit functions are performed such that any associated USE AFTER EXCEPTION procedures are executed. However, such a USE procedure must not execute any statement manipulating file-name-2 or accessing its record area. On the first attempt to write more records than the file can hold, any USE AFTER EXCEPTION procedure specified for the file is executed. When control is returned from the USE procedure, or if such a procedure is not specified, an implicit CLOSE statement without any optional phrases is executed for the file.
9. An input procedure can consist of any statements that select, modify, or copy records made available to it, one at a time, by its RELEASE statement.

The range of statements within an input procedure includes all statements executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements within that procedure, as well as all statements in declarative procedures executed as a result of the execution of statements within that procedure. The range of the input procedure must not cause the execution of any MERGE, RETURN, or SORT statement.
10. If an input procedure is specified, control is passed to it before file-name-1 is sorted. The released records are sorted after control passes the last statement in the procedure. These records are then made available to file-name-1.

11. If the GIVING phrase is specified, all the sorted records are written to filename-3. At the start of execution of the SORT statement, this file cannot be open.

For each file in the phrase

- An implicit OPEN OUTPUT statement is executed for the file.
- The sorted records are written to the file by an implicit WRITE statement without any optional phrases. For relative files, the relative key data item for the first record contains the value 1; for the second record it contains the value 2, and so on. After completion of the SORT statement, the relative key data item has the relative record number of the last record returned to the file.
- An implicit CLOSE statement without any optional phrases is executed for the file.

These implicit functions are performed such that any associated USE AFTER EXCEPTION procedures are executed. However, such a USE procedure must not execute any statement manipulating file-name-3 or accessing its record area. On the first attempt to write more records **than the file can hold, any** USE AFTER EXCEPTION procedure specified for the file is executed. If control is returned from the USE procedure, or if such a procedure is not specified, an implicit CLOSE statement without any optional phrases is executed for the file.

12. If file-name-3 contains only fixed-length records, any record in file-name-1 that contains fewer character positions is space filled on the right when that record is written to file-name-3.
13. An output procedure can consist of any statements that select, modify, or copy the sorted records from file-name-1 that are made available to it, one at a time, by its RETURN statement. The range of the output procedure includes all statements executed as the result of CALL, EXIT, GO TO, and PERFORM statements, as well as all statements in declarative procedures executed as a result of the execution of statements in this range. This range must not cause the execution of any MERGE, RELEASE, or SORT statement.
14. If an output procedure is specified, control passes to it when the sort operation is able to select the next record in sorted order when requested by the RETURN statements in the output procedure. When control passes the last statement in the procedure, the SORT statement ends.

15. During execution of the output procedure (including execution of any USE AFTER EXCEPTION procedure that is implicitly invoked while executing the SORT statement), no statement can be executed that manipulates filename-3 or its record area.
16. The success or failure of a sort operation can be determined by checking the value of the special register RETURN-CODE. Possible values are as follows:

RETURN-CODE	Meaning
0	Success.
4	No records available for sorting <sup>a</sup>
8	Insufficient space in stack or I-O buffer
12	Record size greater than 2,024 bytes
16	Invalid sort key
20	Program check error
24	Input records out of order
28	Input record count in error
<sup>a</sup> Either no input record met the selection criteria, or an empty input file was specified.	

## START Statement for Workstation Files

When used with workstation files, the START statement lets the program determine the current file status of a workstation file. This file status in turn determines whether a subsequent READ request waits for the keyboard to be locked or if it is processed immediately.

### *Format*

START file-name-1

### *Syntax Rule*

File-name-1 must have random access and a device type of DISPLAY.

### *General Rules*

1. File-name-1 must be open in 1-O mode before execution of the START statement.
2. Records in file-name-1 must be variable-length
3. The START statement updates the FILE STATUS data item. If the second byte of the FILE STATUS data item (the Attention Data-name or AID character) is blank, the keyboard is unlocked, and a subsequent read operation must wait for an operator response. If the AID character is not blank, the keyboard is locked and the AID character indicates which key (Enter or a PF key) was last used by the workstation operator. Refer to Appendix E for more information about AID characters.

## START Statement for Relative and Indexed Files

The START statement sets the file position indicator to a point from which subsequent sequential retrieval of records begins.

### **Format**

```
START file-name-1  
    [ KEY [data-name-1] relational-operator data-name-2 ]  
    [ INVALID KEY imperative-statement-1 ]  
    [ NOT INVALID KEY imperative-statement-2 ]  
    [ END-START ]
```

### *Syntax Rules*

#### *Relative and Indexed Files*

1. File-name-1 must have sequential or dynamic access.
2. Data-name-1 and data-name-2 can be qualified
3. The INVALID KEY phrase must be specified if no applicable USE procedure applies to file-name-1.

#### *Relative Files*

4. Data-name-2 must be the relative key data item for the file.
5. Data-name-1 cannot be specified.

#### *Indexed Files*

6. If the KEY phrase is specified, data-name-2 must be a record key (either prime or alternate) for the file or an alphanumeric data item. The data item's leftmost character position in the record corresponds to the leftmost character position of a record key for file-name-1, and its length is not greater than the length of that record key or, if data-name-1 is specified, any data name.
7. Data-name-1 must be an alternate key for the file.

### *General Rules*

#### *Relative and Indexed Files*

1. File-name-1 must be open in INPUT, I-O, or SHARED mode.
2. If the KEY phrase is not specified, the relational operator IS EQUAL TO is implied.
3. Execution of the START statement does not alter either the record area of the file or the contents of the data item specified in the DEPENDING ON phrase in the file.



4. The START statement updates the file position indicator.
5. If, at the time of execution of the START statement, the file position indicator indicates that an optional input file is not present, the INVALID KEY condition exists and the execution of the START statement is unsuccessful.
6. If the operation is successful and the NOT INVALID KEY phrase is specified, execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the START statement. If this phrase is not specified, control passes to the statement following the START.
7. If the START is unsuccessful, the file position indicator is set to indicate that no valid next record has been established. Execution continues according to the rules for exception handling.
8. The END-START phrase delimits the scope of the START statement.

#### *Relative Files*

9. The comparison specified by the relational operator in the KEY phrase is made between a data item of a record in file-name-1 and the relative key data item of the file. (The numeric comparison rules apply.) The file position indicator is set to indicate the first record in the file that satisfies the comparison.

#### *Indexed Files*

10. A key of reference is established. The following rules apply:
  - If the KEY phrase is not specified, the key of reference is the prime record key.
  - If data-name-1 is specified, it is the key of reference.
  - If the KEY phrase is specified without data-name-1, and the FILE-CONTROL entry for file-name-1 specifies integer-4, the key of reference is the prime record key.
  - If the KEY phrase is specified without data-name-1, and the FILE-CONTROL entry for file-name-1 does not specify integer-4, the key of reference is specified by data-name-2, or the key whose leftmost character position is the same as the leftmost position of data-name-2.

11. The comparison specified by the relational operator in the KEY phrase is made between the key of reference and the key items for each record in the file. The following rules apply:
- This comparison is made according to the collating sequence of the file. If the operands are of unequal size, comparison proceeds as though the longer one were truncated on the right such that its length is equal to that of the shorter. All other nonnumeric comparison rules apply. The file position indicator is set to the value of the key of reference of the first record in the file whose key satisfies the comparison.
  - If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists, the START statement is unsuccessful, and no key of reference is established.

## STOP Statement

The STOP statement temporarily or permanently suspends program execution. The STOP literal-1 form of this statement is an obsolete element, scheduled to be deleted from the next revision of Standard COBOL.

### **Format**

$$\underline{\text{STOP}} \left\{ \begin{array}{l} \underline{\text{RUN}} \\ \text{literal-1} \end{array} \right\}$$

### **Syntax Rules**

1. Literal-1 cannot be a figurative constant that begins with the word ALL.
2. If a STOP RUN statement is in a consecutive sequence of imperative statements in a sentence, it must be the last statement in that sequence.
3. Literal-1 can be nonnumeric or numeric; if numeric, it must be an unsigned integer.
4. If literal-1 is a figurative constant, it cannot be a user-defined figurative constant.

### **General Rules**

1. If STOP RUN is specified execution of the program ceases, and control is transferred back to the operating system. During execution of a STOP RUN statement, an implicit CLOSE statement (without any optional phrases) is executed for all open files in the run unit. Declarative USE procedures associated with these files are not executed.

The program can put a value into the special register RETURN-CODE. STOP RUN passes the value of RETURN-CODE to the operating system. RETURN-CODE has the attributes of a PICTURE S9(9) USAGE BINARY data item.

2. STOP literal-1 is an archaic form and identical in function to a DISPLAY literal-1 statement.

## STRING Statement

The STRING statement concatenates (that is, strings) two or more data items, in whole or in part, into a single data item.

### **Format**

```
STRING { { data-name-1 } ... DELIMITED BY { data-name-2 } } ...  
      { literal-1 } SIZE } ...  
      INTO data-name-3  
      [ WITH POINTER data-name-4 ]  
      [ ON OVERFLOW imperative-statement-1 ]  
      [ NOT ON OVERFLOW imperative-statement-2 ]  
      [ END-STRING ]
```

### **Syntax Rules**

1. All literals must be nonnumeric and can be any figurative constant without the word ALL.
2. All data-names, except data-name-4, must have a usage of DISPLAY.
3. If data-name-1 or data-name-2 is numeric, it must not contain a P in its PICTURE.
4. Data-name-3 cannot be edited, have a JUSTIFIED clause in its description, or be reference modified.
5. Data-name-4 must be a numeric integer that does not contain a P in its PICTURE. It must be capable of containing a value equal to 1 plus the size of data-name-3.

### **General Rules**

1. Data-name-1 or literal-1 is the sending item; data-name-2, the receiving item.
2. Literal-2 or data-name-2 indicates the delimiter. If a figurative constant is used as the delimiter, it is taken as a single-character, nonnumeric literal.
3. Literal-1 or literal-2 is a figurative constant; it is taken as a 1-character data item with a usage of DISPLAY.
4. The STRING statement transfers data from the sending item to the receiving item in accordance with the rules for alphanumeric to alphanumeric moves, except that no space filling is provided.
5. If the DELIMITED phrase is specified without the SIZE phrase, the contents of the sending field are transferred to the receiving field in the sequence specified in the STRING statement. The transfer begins with the leftmost character of the sending field and proceeds from left to right, until either the end of the sending or receiving item is reached or the character(s) specified by literal-2 or data-name-2 is found in the sending item. (The characters in literal-2 or data-name-2 are not transferred.)

6. If DELIMITED BY SIZE is specified, the entire contents of each sending field are transferred in the sequence specified in the STRING statement. This continues until all data has been transferred or until the end of the receiving field is reached.
7. If the POINTER phrase is specified, data-name-4 must be set to a value greater than zero prior to the execution of the STRING statement. If the POINTER phrase is not specified, operation proceeds as shown in the following list, assuming a hypothetical data-name-4 had been set to an initial value of 1:
  - Before each move of a character to the receiving item, the value of data-name-4 is checked to see that it is not less than one or greater than the number of character positions in data-name-3. If it is outside these limits, no data is transferred.
  - If data-name-4 is within the preceding limits, a character is moved from the sending item into the position of the receiving item designated by the current value of data-name-4.
  - The value of data-name-4 is incremented by one after the character is moved, and assuming the new value is within the above limits, another character is moved as described previously.
8. The STRING statement changes only that portion of the receiving item that was referenced. All other parts of the item contain the data that was present before the execution of the STRING statement.
9. If the value of data-name-4 exceeds the number of positions in the receiving item, the OVERFLOW condition exists. If the ON OVERFLOW phrase is specified, execution continues with each statement in imperative-statement-1. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-1, control is transferred to the end of the STRING statement.
10. If the OVERFLOW condition does not occur, following the moving of the last character into the receiving item, and if the NOT ON OVERFLOW phrase is specified, execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the STRING statement.
11. If the storage areas of data-name-1 or data-name-2 overlap the storage areas of data-name-3 or data-name-4, or if the storage areas of data-name-3 and data-name-4 overlap, the SEARCH statement does not operate properly.

## SUBTRACT Statement

The SUBTRACT statement subtracts one item or the sum of two or more items from one or more items and sets one or more items equal to the result. Refer to Chapter 6 for details about the SIZE ERROR and NOT SIZE ERROR clauses.

### Format 1

```
SUBTRACT { identifier-1  
           literal-1 } ...  
           FROM { data-name-2 [ ROUNDED ] } ...  
[ ON SIZE ERROR imperative-statement-1 ]  
[ NOT ON SIZE ERROR imperative-statement-2 ]  
[ END-SUBTRACT ]
```

### Format 2

```
SUBTRACT { identifier-1  
           literal-1 } ... FROM { identifier-2  
                                   literal-2 }  
           GIVING { identifier-3 [ ROUNDED ] } ...  
[ ON SIZE ERROR imperative-statement-1 ]  
[ NOT ON SIZE ERROR imperative-statement-2 ]  
[ END-SUBTRACT ]
```

### Format 3

```
SUBTRACT CORRESPONDING identifier-1  
           FROM identifier-2 [ ROUNDED ]  
[ ON SIZE ERROR imperative-statement-1 ]  
[ NOT ON SIZE ERROR imperative-statement-2 ]  
[ END-SUBTRACT ]
```

### ***Syntax Rules***

1. Each data-name must be a numeric item, except that in format 2 a data-name following the word GIVING can be a numeric item or a numeric edited item; in format 3, each data-name must be a group item.
2. Each literal must be numeric.
3. The composite of operands cannot contain more than 18 digits. In format 1, the composite of operands is formed from all the operands in the statement. In format 2, the composite of operands is formed from all the operands in the statement that precede the word GIVING. In format 3, the composite of operands is determined by each pair of corresponding data items.
4. CORR can be used instead of CORRESPONDING.

### ***General Rules***

1. In a format 1 SUBTRACT, the values of the operands preceding the word FROM are added together and their sum is stored in a temporary data item. This temporary item is then subtracted from each data item following the word FROM, the result being stored as the new value of that data-name.
2. In a format 2 SUBTRACT, the values of the operands preceding the word FROM are added together. Their sum is then subtracted from literal-2 or data-name-2 and the result of the subtraction is stored as the new value of each data item following GIVING.
3. In a format 3 SUBTRACT, the data items of data-name-1 are subtracted from their corresponding items in data-name-2. The results of each subtraction are the new contents of the item in data-name-2.
4. If no size error occurs and the NOT SIZE ERROR phrase is not specified, program execution continues with a statement following the SUBTRACT. If the NOT SIZE ERROR phrase is specified, execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the SUBTRACT statement.
5. If a size error occurs and the SIZE ERROR phrase is not specified, the contents of the affected data items are incorrect. If the SIZE ERROR phrase is specified, the contents of the affected data items are the same as they were at the start of the operation. Program execution continues with each statement in imperative-data-name-1. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-data-name-1, control is transferred to the end of the SUBTRACT statement.

## UNSTRING Statement

The UNSTRING statement moves data from a single field to multiple fields.

### **Format**

```
UNSTRING data-name-3  
[ DELIMITED BY [ALL] { data-name-2  
    literal-2 }  
[ OR [ALL] { data-name-3  
    literal2 } ] ... ]  
INTO {data-name-4 [DELIMITER IN data-name-5]  
    [COUNT IN data-name-6]}...  
[ WITH POINTER data-name-7 ]  
[ TALLYING IN data-name-8 ]  
[ ON OVERFLOW imperative-statement-1 ]  
[ NOT ON OVERFLOW imperative-statement-2 ]  
[ END-UNSTRING ]
```

### Syntax Rules

1. Each literal must be nonnumeric and can be any figurative constant without the word ALL.
2. No data-name can be a level 88 entry.
3. Data-name-1 cannot be reference modified.
4. Data-name-1, data-name-2, data-name-3, and data-name-5 must be alphanumeric.
5. Data-name-4 can be alphabetic, alphanumeric, or numeric; but if it is numeric its PICTURE string cannot contain a P. It must have a usage of DISPLAY.
6. Data-name-6 and data-name-8 must be integers that do not have a P in their PICTURE strings.
7. Data-name-7 must be an integer of sufficient size to contain a value equal to 1 plus the size of data-name-1. Its PICTURE string cannot have a P.
8. The DELIMITER IN phrase and the COUNT IN phrase can be specified only if the DELIMITED BY phrase is also specified.

### **General Rules**

1. Data-name-1 is the sending item; data-name-4, the receiving item.
2. Each receiving item can have two other items associated with it: data-name-5 is the delimiter receiving item, and data-name-6 contains the count of the number of



characters in data item-1 that were moved to data-name-4. (This count does not include any delimiter character.)

3. Literal-1, literal-2, or the corresponding data items are delimiters. A delimiter can contain any character in the computer's character set. The sending item is checked for delimiters in the order in which they are specified in the UNSTRING statement. If a delimiter contains two or more characters, all of these characters must be present in contiguous positions in the sending item to be considered an occurrence of the delimiter. If a figurative constant is used as a delimiter, it is taken as a single-character, nonnumeric literal.
4. Data is moved from data-name-1 to the first data-name-4. Characters moved are treated as an elementary alphanumeric data item and are moved according to the rules for the MOVE statement.

Note: Because a given character in data-name-1 can be moved as data, moved as a delimiter, or skipped over as a delimiter, the term "examined" means that the character has been subjected to one of these three operations.

If the POINTER phrase is not specified, the move begins with the leftmost character of data-name-1. If the POINTER phrase is specified, the move begins with the character in data-name-1 whose position is indicated by dataname-7.

If the DELIMITED BY phrase is not specified, the number of characters moved is equal to the size of the current receiving item. However, if the sign of the receiving item is defined as occupying a separate character position, the number of characters examined is one less than the size of the current receiving item.

If the DELIMITED BY phrase is specified, the move stops when an instance of any of the delimiters is found. However, if the last character of dataname-1 is encountered before any delimiters are found, the operation terminates.

After the data is transferred to data-name-4., and if there are further receiving items and the last character of data-name-1 has not yet been examined, the UNSTRING operation repeats. In this case, if the DELIMITED BY phrase is not specified, the next move begins with the character to the right of the last character moved. If the DELIMITED BY phrase is specified, the next move begins with the first character to the right of the delimiter that was found. This sequence of operations continues until all the characters in data item-1 have been examined or there are no more receiving items.

5. If the DELIMITER IN phrase is specified when a delimiter is found, it is treated as an elementary alphanumeric data item and moved into the associated data-name-5 according to the rules for the MOVE statement. If no delimiter is found, the associated data-name-5 is filled with spaces.
6. If the ALL phrase is specified multiple contiguous instances of a delimiter are treated as if they were one instance, and only this one instance of the delimiter is moved to the delimiter receiving item. If the ALL phrase is not specified, when two contiguous instances of a delimiter are encountered, and if the current receiving item is alphabetic or alphanumeric, it is filled with spaces. If the receiving item is numeric, it gets the value zero.
7. If two or more delimiters are specified, and if any one of them occurs in the sending item, the character(s) in the sending field is considered to be a single delimiter. No character(s) in the sending item is ever considered part of more than one delimiter.

8. If the COUNT IN phrase is specified, a value equal to the number of characters moved (excluding any delimiter characters) is put into data-name-6.
9. Data-name-7 and data-name-8 are associated with the overall operation of the statement, and not with a particular receiving item. *The program must initialize these data items before the start of the UNSTRING operation.* Data-name-7 indicates the character position in data-name-1 at which the data transfer begins. Data-name-8 is incremented by 1 for each data-name-4 accessed during the UNSTRING operation, so at the end of the operation it indicates *how many* items have had data transferred to them (useful information when the sending item is smaller than the combined receiving items).
10. Data-name-7 is incremented by one for each character examined in data-name-1. At the end of the UNSTRING operation, the value of data-name-7 is equal to its initial value plus the number of characters examined in data-name-1.
11. If at the start of the UNSTRING operation the value in data-name-7 is less than 1 or greater than the size of data-name-1, or if during execution, all receiving items have been acted upon and data-name-1 still contains characters that have not been moved, an overflow condition exists.
12. If an overflow condition exists, the UNSTRING operation stops, and if the ON OVERFLOW phrase is specified, execution continues with each statement in imperative-statement-1. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-1, control is transferred to the end of the UNSTRING statement. If the ON OVERFLOW phrase is not specified, control passes to the end of the UNSTRING statement.
13. If the overflow condition does not occur when the UNSTRING operation is over, and if the NOT ON OVERFLOW phrase is specified, execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the UNSTRING statement. If the NOT ON OVERFLOW phrase is not specified, execution continues with the statement following the UNSTRING statement.
14. If the TALLYING phrase is specified when execution of the UNSTRING statement is completed, the value of data item-8 is equal to its initial value plus the number of receiving data items accessed.
15. The unstring operation does not operate properly if
  - The storage area of data-name-1, data-name-2, or data-name-3 overlaps the storage area of data-name-4, data-name-5, data-name-6, data-name-7, or data-name-8.
  - The storage area of data-name-4, data-name-5, or data-name-6 overlaps the storage area of data-name-7 or data-name-8.
  - The storage area of data-name-7 overlaps the storage area of data-name-8.
16. The END-UNSTRING phrase delimits the scope of the UNSTRING statement.

## USE Statement

The USE statement specifies input-output error-handling procedures that are executed in addition to the standard procedures provided by the file system.

### Format 1

$$\text{USE [GLOBAL] AFTER STANDARD } \left\{ \begin{array}{l} \text{ERROR} \\ \text{EXCEPTION} \end{array} \right\} \text{ PROCEDURE}$$
  
$$\text{ON } \left\{ \begin{array}{l} \text{\{file-name-1\}...} \\ \text{INPUT} \\ \text{OUTPUT} \\ \text{I-O} \\ \text{EXTEND} \\ \text{SHARED} \\ \text{SPECIAL-INPUT} \end{array} \right\}$$

### Format 2

USE [GLOBAL] AFTER DEADLOCK

### Syntax Rules

#### Both Formats

1. The USE statement itself is never executed; it merely defines the conditions under which the associated procedure is executed
2. A USE statement must immediately follow a section header in the Declaratives section of the Procedure Division and must be in a sentence by itself. The remainder of the section can consist of zero, one, or more paragraphs that define the procedures to be used.
3. The words ERROR and EXCEPTION are equivalent.

#### Format 1

1. If the name of a file is written in a USE statement, that file is said to be *explicitly specified*. If an OPEN mode is written in a USE statement, a file in that mode is said to be *implicitly specified*.
2. A file can be explicitly specified in no more than one USE statement.
3. Files implicitly or explicitly specified in a USE statement need not have all the same organization or access.
4. The INPUT, OUTPUT, I-O, SHARED, SPECIAL-INPUT, and EXTEND phrases can each be specified only once in the declarative portion of a given Procedure Division.

## ***General Rules***

### ***Both Formats***

1. Declarative procedures can be included in any source program. The declarative is invoked when any of the conditions described in the associated USE statement occur.
2. In a separately compiled program, the only declarative that can be invoked is the one in that program.
3. If programs are nested, more than one program can have an applicable declarative. In this case, the declaratives are examined as follows, the first qualifying declarative being the one that is executed:
  - If there is a declarative in the program that contains the statement that caused the condition, it is executed.
  - If there is a declarative with a GLOBAL phrase in the program that directly contains the program that caused the condition, it is executedIf not, this second criterion is applied to each inclusive containing program until either a declarative is found or the outermost program is reached
4. A declarative procedure can be referenced in a different declarative procedure. If referenced from a nondeclarative procedure, it must be by means of a PERFORM statement.
5. A declarative procedure cannot reference a nondeclarative procedure.
6. A USE procedure must not cause the execution of any statement that would cause the execution of an active USE procedure. That is, only one USE procedure can be active at any given time.

### ***Format 1***

7. When an input-output operation fails, the associated USE procedures are executed by the file system after completion of its standard error-handling routines. However, if an INVALID KEY or AT END phrase is specified in the offending statement, it takes precedence. The following rules apply:
  - If a file-name is explicitly specified, the associated procedure is executed when the exception condition occurs. In addition, no other USE statement is considered to apply to that file.
  - If INPUT, OUTPUT, I-O, SHARED, SPECIAL-INPUT, or EXTEND is specified, the associated procedure is executed when the exception condition occurs for any file open or in the process of being opened in the specified mode, except those files that are explicitly specified in another USE statement.
8. After execution of a USE procedure, control is transferred to the invoking routine. If the file status data item does not indicate a critical input-output error, control passes to the next executable statement following the statement that caused the exception.

*Format 2*

9. The USE AFTER DEADLOCK statement specifies a deadlock exit in a transaction-capable environment. If a deadlock occurs, the original program, by means of the declarative, may be able to retain control instead of being canceled.
10. Within the USE procedure, an exit procedure can be specified by a GO TO statement. After execution of the USE procedure, control is returned to the user specified exit procedure; if there is no exit procedure, the program is canceled.
11. If a deadlock occurs and the system deadlock handler fails (for example, it is unable to perform a ROLLBACK), the program is canceled.

## WRITE Statement for Sequential Files

The WRITE statement writes a record to a file.

### Format

```
WRITE record-name-1 [ FROM identifier-1 ]  
  
[ { BEFORE } ADVANCING { mnemonic-name  
  AFTER } { PAGE  
  user-figurative-constant-1  
  identifier-2 } [ LINE  
  integer-1 ] [ LINES ]  
  
[ AT { END-OF-PAGE } imperative-statement-1  
  EOP ]  
  
[ NOT AT { END-OF-PAGE } imperative-statement-2  
  EOP ]  
  
[ END-WRITE ]
```

### Syntax Rules

1. The storage areas of record-name-1 and data-name-1 cannot overlap.
2. Record-name-1 can be qualified.
3. If the file has a LINAGE clause, the ADVANCING phrase cannot specify a user-figurative-constant.
4. Data-name-2 must be an integer.
5. Integer-1 must be positive or zero.
6. User-figurative-constant-1 must be defined as a 2-byte data item in the FIGURATIVE-CONSTANTS paragraph of the Environment Division.
7. The ADVANCING PAGE phrase and the END-OF-PAGE phrase cannot both be specified in the same WRITE statement.
8. If either the END-OF-PAGE or the NOT END-OF-PAGE phrase is specified, the file must have a LINAGE clause.
9. Mnemonic-name must be the user-defined word that is associated with ONELINE in the SPECIAL-NAMES paragraph.
10. The terms END-OF-PAGE and EOP are equivalent.

### General Rules

1. File-1 must be open in the OUTPUT, I-O, or EXTEND mode.

2. The WRITE statement updates the FILE-STATUS data item. It does not affect the file position indicator.
3. The relationship of records in a sequential file is established by the order in which they were written to the file when the file was created. This relationship does not change except when records are added to the end of the file.
4. A WRITE statement for a sequential file in EXTEND mode adds records to the file as though the file were opened in OUTPUT mode. If there are already records in the file, the first record written after the execution of the OPEN statement with the EXTEND phrase is the successor of the previous last record in the file.
5. The number of character positions in record-1 must not be larger than the largest or smaller than the smallest number of character positions allowed by the RECORD IS VARYING clause for its file. If either of these limits is violated, the write operation does not take place, and the WRITE statement is unsuccessful.
6. If the WRITE is successful, that is, if no exception condition exists, the rewritten record is no longer available in the file's record area. However, if the file is specified in a SAME RECORD AREA clause, the record is available both in the file's record area and also as a record of any other file specified in that clause that is currently open in OUTPUT mode.
7. If the WRITE statement is unsuccessful for any reason, the file's record area is unaffected, and the FILE STATUS data item is set to indicate the reason for the failure.
8. An error condition occurs when an attempt is made to write more records than the file can hold. In this case, the contents of the record area are unaffected, and the file status data item is set to indicate a boundary violation. Then, if there is an applicable USE procedure, it is executed. If there is no USE procedure, the program is terminated.
9. If the end of a reel (or unit) is recognized and there has been no boundary violation, the standard ending reel label procedure is done; a reel/unit swap is made, the current volume pointer being updated to point to the next reel for the file; and then a standard beginning reel label procedure is performed,
10. If the statement has the FROM phrase, the record is moved from data item-1 to the file's record area according to the rules for the MOVE statement. After the WRITE has completed, the record is available in data item-1, even if it is not available in the file's record area. Any subscripting associated with dataname-1 is evaluated before the data is moved to the record area.
11. The END-WRITE phrase delimits the scope of the WRITE statement.
12. If the ADVANCING phrase is specified and integer- I or data-name-2 is positive, the page is advanced the appropriate number of lines. If the value is zero, no advancing takes place. If the value is negative, the program is in error. If a user-figurative-constant is specified, advancing is controlled by the 2-byte write control character that must have been defined in the FIGURATIVE-CONSTANTS paragraph. If bit 0 of the first byte is 0, bits 1 through 7 of the second byte indicate the number of lines to be skipped. If bit 0 of the first byte is 1, the second byte indicates a top-of-form (HEX "01") or vertical tab (HEX "02").

13. If the BEFORE phrase is specified, the line is printed before the page is advanced. If the AFTER phrase is specified, the line is printed after the page is advanced.
14. If PAGE is specified, the line is printed BEFORE or AFTER the printer is positioned to the first line of the page body on the next page (as specified by the LINAGE clause). If the file does not have a LINAGE clause, and if the device type is PRINTER, the line is printed on the page BEFORE or AFTER the printer is positioned to the next page. If the device is not PRINTER, advancing takes place as if BEFORE or AFTER ADVANCING 1 LINE had been specified.
15. If a mnemonic-name is specified, it causes the advancement of one line.
16. The END-OF-PAGE condition occurs when execution of a WRITE statement with the END-OF-PAGE phrase causes the LINAGE-COUNTER to equal or exceed the value specified in the LINAGE clause. This causes printing or spacing within the footing area. When the END-OF-PAGE condition occurs, and if the END-OF-PAGE clause is specified, the WRITE statement is executed, and then execution continues with each statement in imperative-statement-1. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-1, control is transferred to the end of the WRITE statement.
17. If the END-OF-PAGE condition does not occur, and the NOT END-OF-PAGE phrase is specified, control is transferred to imperative-statement-2. If the WRITE statement was successful, this transfer occurs after the record has been written and the associated FILE STATUS data item has been updated. If the WRITE statement was unsuccessful, the transfer occurs after the associated FILE STATUS data item has been updated and any applicable USE procedure has been executed  
  
After the transfer, execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the WRITE statement.
18. The PAGE OVERFLOW condition occurs when the result of a WRITE statement (with or without an END-OF-PAGE phrase) would not be able to be fully accommodated within the current page body. (That is, when the execution of the statement would cause LINAGE-COUNTER to exceed the value specified in the LINAGE clause.) The PAGE OVERFLOW condition also occurs when execution of a WRITE statement would cause LINAGECOUNTER to simultaneously exceed both the page body and the footing area (as specified by integer-1 and integer-2 in the LINAGE clause).
19. If the PAGE OVERFLOW condition occurs, the record is written on the logical page BEFORE or AFTER the device is repositioned to the first line that can be written on the next logical page. If the END-OF-PAGE phrase is specified, imperative-statement-1 is executed after the record has been written and the device repositioned. Execution continues with each statement in imperative-statement-1. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-1, control is transferred to the end of the WRITE statement.



## WRITE Statement for Relative Files

The WRITE statement writes a record to a file.

### **Format**

```
WRITE record-name-1 [ FROM data-name-1 ]  
    [ INVALID KEY imperative-statement-1 ]  
    [ NOT INVALID KEY imperative-statement-2 ]  
    [ END-WRITE ]
```

### **Syntax Rules**

1. The storage areas of record-name-1 and data-name-1 cannot overlap.
2. Record-name-1 can be qualified.
3. If there is no applicable USE procedure for the associated file, the INVALID KEY phrase must be specified.

### **General Rules**

1. File-name-1 must be open in the OUTPUT, I-O, or EXTEND mode.
2. The WRITE statement updates the FILE-STATUS data item. It does not affect the file position indicator.
3. If the WRITE statement is unsuccessful for any reason, the file's record area is unaffected, and the FILE STATUS data item is set to indicate the reason for the failure.
4. If the WRITE is successful, that is, if no EXCEPTION condition exists, the written record is no longer available in the file's record area. However, if the file is specified in a SAME RECORD AREA clause, the record is available both in the file's record area and also as a record of any other file specified in the clause that is currently open in the OUTPUT mode. If the NOT INVALID KEY phrase is not specified, control passes to the end of the WRITE statement. If it is specified, execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the WRITE statement.
5. If the INVALID KEY condition occurs and the INVALID KEY phrase is specified, execution continues with each statement in imperative-statement-1. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-1, control is transferred to the end of the WRITE statement. Any associated USE procedure is not executed. If the INVALID KEY phrase is not specified, the applicable USE procedure is executed, and control returns to the next executable statement following the WRITE.

6. If an EXCEPTION condition other than an INVALID KEY condition occurs and there is an applicable USE statement, it is executed, and upon its completion control returns to the statement following the WRITE. If a USE procedure is not specified, the program is terminated.
7. The number of character positions in data-name-1 must not be larger than the largest or smaller than the smallest number of character positions allowed by the RECORD IS VARYING clause for its file. If either of these limits is violated, the write operation does not take place, and the WRITE statement is unsuccessful.
8. If the statement has the FROM phrase, the record is moved from dataname-1 to the file's record area according to the rules for the MOVE statement. After the WRITE has completed, the record is available in dataname-1, even if it is not available in the file's record area. Any subscripting associated with data-name-1 is evaluated before the data is moved to the record area.
9. The END-WRITE phrase delimits the scope of the WRITE statement.

## WRITE Statement for Relative and Indexed Files

The WRITE statement writes a record to a file.

### Format

```
WRITE record-name-1 [ FROM data-name-1 ]  
    [ TIMEOUT OF { data-name-2 } [ SECOND ]  
      { integer-1 } [ SECONDS ]  
      [ HOLDER-ID IN data-name-3 ]  
      { imperative-statement-1 } ]  
      NEXT SENTENCE  
    [ INVALID KEY imperative-statement-2 ]  
    [ NOT INVALID KEY imperative-statement-3 ]  
    [ END-WRITE ]
```

### Syntax Rules

1. The storage areas of record-name-1 and data-name-1 cannot overlap.
2. Record-name-1 can be qualified
3. The TIMEOUT clause can be specified only if the file associated with record-name-1 has indexed organization.
4. Data-name-2 must be an integer data item. Integer-1 must be from 0 to 255.
5. Data-name-3 must be defined in the Working-Storage section or Linkage section and have a PICTURE of X(3).
6. If there is no applicable USE procedure for the associated file, the INVALID KEY phrase must be specified

### General Rules

1. File-name-1 must be open in the OUTPUT, I-O, SHARED, or EXTEND mode.
2. The WRITE statement updates the FILE STATUS data item. It does not affect the file position indicator.
3. The data item specified as the prime record key must be set by the program to the value of the key of the record being written prior to the execution of the WRITE statement. The value of the prime record key must be unique within the records in the file.
4. If a Relative file is in OUTPUT mode, and the access is sequential, the first record to be written has a relative record number of 1 and subsequent records have relative record numbers of 2, 3, 4, etc. If the file has a relative key data item, the relative record number of the record that was written is placed into this item. If the access is random or dynamic, the value of the relative key data item must be set to the relative record number of the record to be written.

5. If a Relative file is in EXTEND mode, the access must be sequential. The first record released to the file system has a relative record number that is one greater than the highest relative record number existing in the file. Subsequent records released to the file system have consecutively higher relative record numbers. If the file has a relative key data item, the relative record number of the record that was written is put into that item.
6. If a Relative file is open in I-O mode, the access must be random or dynamic. The value of the relative key data item must be set to the relative record number of the record to be written.
7. The INVALID KEY condition exists for a Relative file (1) if the access is random or dynamic, and the relative key data item specifies a record that is already in the file; (2) if an attempt is made to write more records than the file can hold; or (3) if the relative record number is too large to fit into the relative key data item. If the INVALID KEY condition occurs, the WRITE statement is unsuccessful.
8. If an Indexed file is in OUTPUT mode and the access is sequential, records must be written in ascending order of prime record key values. If the access is random or dynamic, records can be written in any order of prime key values.
9. If an Indexed file is in EXTEND mode, the access must be sequential. Each record written must have a prime record key whose value is greater than the highest prime record key value already in the file.
10. If an Indexed file is in I-O mode, the access must be random or dynamic. Records can be written in any order of prime key values.
11. The INVALID KEY condition exists for an Indexed file (1) if the ACCESS mode is sequential and the value of the prime record key is not greater than the value of the prime record key of the previous record; (2) if the file is in the OUTPUT, SHARED, or I-O mode and the value of the prime record key equals the value of the prime record key of a record already existing in the file; (3) if the file is opened in the OUTPUT, EXTEND, or I-O mode and the value of an alternate record key for which duplicates are not allowed equals the corresponding data item of a record already existing in the file, or an attempt is made to write more records than the file can hold.
12. If the INVALID KEY condition occurs and the INVALID KEY phrase is specified, execution continues with each statement in imperative-statement-2. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-2, control is transferred to the end of the WRITE statement. Any associated USE procedure is not executed. If the INVALID KEY phrase is not specified, the applicable USE procedure is executed, and control returns to the next executable statement following the WRITE.
13. If an exception condition other than an INVALID KEY condition occurs, and if there is an applicable USE statement, it is executed, and upon its completion control returns to the statement following the WRITE. If a USE procedure is not specified, the program is terminated
14. The number of character positions in record-name-1 must not be larger than the largest or smaller than the smallest number of character positions allowed by the

RECORD IS VARYING clause for its file. If either of these limits is violated, the write operation does not take place, and the WRITE statement is unsuccessful.

15. If the TIMEOUT phrase is specified and the write operation cannot be completed in the number of seconds specified by data-name-1 or integer-1, imperative-statement-1 is executed. If zero seconds is specified, the timeout exit is taken immediately if the write operation cannot be completed. If the HOLDER-ID phrase is specified in the TIMEOUT phrase, the logon initials of the user currently holding the desired resource is put into data-name-2.
16. If the WRITE statement is unsuccessful for any reason, the file's record area is unaffected, and the FILE STATUS data item is set to indicate the reason for the failure.
17. If the WRITE is successful, that is, if no exception condition exists, the written record is no longer available in the file's record area. However, if the file is specified in a SAME RECORD AREA clause, the record is available both in the file's record area and also as a record of any other file specified in the clause that is currently open in the OUTPUT mode. If the NOT INVALID KEY phrase is not specified, control passes to the end of the WRITE statement. If it is specified, execution continues with each statement in imperative-statement-3. If one of these statements causes an explicit transfer of control, it is done in accordance with the rules for that statement. Otherwise, upon completion of imperative-statement-3, control is transferred to the end of the WRITE statement.
18. If the statement has the FROM phrase, the record is moved from data-name-1 to the file's record area, according to the rules for the MOVE statement. After the WRITE has completed, the record is available in data-name-1, even if it is not available in the file's record area. Any subscripting associated with data-name-1 is evaluated before the data is moved to the record area.
19. The END-WRITE phrase delimits the scope of the WRITE statement.

# Chapter 8 Intrinsic Functions

## Introduction

The Intrinsic Functions provide the capability to reference a data item whose value is derived automatically at the time of reference during the execution of the object program. A function is a temporary data item whose value is determined by invoking a mechanism provided by the implementor at the time the function is referenced during the execution of a statement.

## Language Concepts

The following sections describe the language concepts.

### Function-Name

A function-name names a mechanism provided by the implementor to determine the value of a function. A function-name is a COBOL word that is taken from a specified list.

### Value Returned by a Function

The value returned by a function is considered to be a data value. A mechanism is provided at object time to assign a data value to a -function when it is referenced. In order to determine the function's value, the evaluation mechanism may require access to data values provided by the referencing program. These data values are provided by specifying parameters, known as arguments, when referencing the function. Specific functions can place constraints on these arguments, such as range, etc. If, at the time a function is referenced, the arguments specified for that reference do not have values that comply with the specified constraints, the returned value for the function is undefined.

A Wang extension provides a `USE AFTER FUNCTION ERROR` declarative to handle such exceptions. If such a declarative is present in a program, it will be executed for these exceptions. Upon return from the declarative procedure statement, execution continues with the undefined value of the function. If no such declarative is present, program execution is terminated with an informative error message.

### Function-Identifier

A function-identifier is used by the programmer to reference a function within the Procedure Division of a COBOL source program. A function-identifier is composed of the keyword `FUNCTION` followed by the function-name and arguments.

## General Description

The following sections provide a general description.

### Arguments

Arguments specify values used in the evaluation of a function. Arguments are specified in the function-identifier. These arguments can be specified as identifiers, as arithmetic expressions, or as literals. The definition of a function specifies the number of arguments required, which can be zero, one, or more. For some functions, the number of arguments that can be specified may be variable. The order in which arguments are specified in a function-identifier determines the interpretation given to each value in arriving at the function value.

Arguments may be required to have a certain class or a subset of a certain class. The types of argument are

**Numeric**- An arithmetic expression must be specified. The value of the arithmetic expression, including the operational sign, is used in determining the value of the function.

**Alphabetic** - An elementary data item of the class alphabetic or a nonnumeric literal containing only alphabetic characters must be specified. The size associated with the argument can be used in determining the value of the function.

**Alphanumeric**- A data item of the class alphabetic or alphanumeric or a nonnumeric literal must be specified. The size associated with the argument can be used in determining the value of the function.

**Integer** - An arithmetic expression that will always result in an integer value must be specified. The value of the arithmetic expression, including the operational sign, is used in determining the value of the function.

The rules for a function can place constraints on the permissible values for arguments in order to permit meaningful determination of the function's value. If, at the time a function is referenced, the arguments specified for that reference do not have values within the permissible range, the returned value for the function is undefined.

A Wang extension provides a `USE AFTER FUNCTION ERROR` declarative to handle such exceptions. If such a declarative is present in a program, it will be executed for these exceptions. Upon return from the declarative procedure statement, execution continues with the undefined value of the function. If no such declarative is present, program execution is terminated with an informative error message.

When the definition of a function permits an argument to be repeated a variable number of times, a table may be referenced by specifying the data-name and any qualifiers that identify the table, followed immediately by subscripting where one or more of the subscripts is the word `ALL`.

When `ALL` is specified as a subscript, the effect is as if each table element associated with that subscript position were specified. The order of the implicit specification of each

occurrence is from left to right, with the first (or leftmost) specification being the identifier with each subscript specified by the word ALL replaced by one, the next specification being the same identifier with the rightmost subscript specified by the word ALL incremented by one.

This process continues with the rightmost ALL subscript being incremented by one for each implicit specification until the rightmost ALL subscript has been incremented through its range of values. If there are any additional ALL subscripts, the ALL subscript immediately to the left of the rightmost ALL subscript is incremented by one, the rightmost ALL subscript is reset to one, and the process of varying the rightmost ALL subscript is repeated. The ALL subscript to the left of the rightmost ALL subscript is incremented by one through its range of values. For each additional ALL subscript, this process is repeated in turn until the leftmost ALL subscript has been incremented by one through its range of values.

If the All subscript is associated with an OCCURS DEPENDING ON clause, the range of values is determined by the object of the OCCURS DEPENDING ON clause. The evaluation of an ALL subscript should result in at least one argument. A Wang extension allows empty ALL subscript selectors, as long as the resultant argument list at the time of reference does not violate the argument constraints of the function.

## Types of Functions

Functions are elementary data items with alphanumeric, numeric, or integer values. Functions cannot be receiving operands. The types of functions are

**Alphanumeric functions**-These are of the class and category alphanumeric. The number of character positions in this data item is specified in the function definition. Alphanumeric functions have an implicit usage of DISPLAY.

**Numeric functions** - These are of the class and category numeric. A numeric function is always considered to have an operational sign. Those characteristics of the returned value not otherwise specified for a given function are defined by the implementor.

The following rules apply:

- A numeric function can be used only in an arithmetic expression.
- A numeric function cannot be referenced where an integer operand is required, even though a particular reference can yield an integer value.

**Integer functions**- These are of the class and category numeric. An integer function is always considered to have an operational sign. Those characteristics of the returned value not otherwise specified for a given function are defined by the implementor. The following rules apply:

- An integer function can be used only in an arithmetic expression.
- An integer function can be referenced where an integer operand is required and where a signed operand is allowed.



## Returned Value

The definition of a function identifies

- For alphanumeric functions, the size of the returned value
- For numeric and integer functions, the sign of the returned value and whether the function is integer
- For some other cases, the value returned

## Date Conversion

The Gregorian calendar is used in the date conversion functions. The starting date of Monday, January 1, 1601, was chosen to establish a simple relationship between the Standard Date and DAY-OF-WEEK; that is, integer date 1 was a Monday, DAY-OF-WEEK 1.

## Function Definitions

Function definitions are described in alphabetical order in the following sections.

### ACOS

The ACOS function returns a numeric value, expressed in radians, that approximates the arccosine of argument-1. The type of this function is floating point.

#### Format

FUNCTION ACOS(argument-1)

#### Arguments

1. Argument-1 must be class numeric.
2. The value of argument-1 must be greater than or equal to -1 and less than or equal to + 1.

#### *Returned Value*

The returned value is the approximation of the arccosine of argument-1 and is greater than or equal to zero and less than or equal to pi.

## ANNUITY

The ANNUITY function (annuity immediate) returns a numeric value that approximates the ratio of an annuity paid at the end of each period for the number of periods specified by argument-2 to an initial investment of one. Interest is earned at the rate specified by argument-1 and is applied at the end of the period, before the payment. The type of this function is floating point.

### Format

FUNCTION ANNUITY (argument-1 argument-2)

### Arguments

1. Argument-1 must be class numeric.
2. The value of argument-1 must be greater than or equal to zero.
3. Argument-2 must be a positive integer.

### Returned Values

1. When the value of argument-1 is zero, the value of the function is the approximation of

$$1/\text{argument-2}$$

2. When the value of argument-1 is not zero, the value of the function is the approximation of

$$\text{argument-1} / (1 - (1 + \text{argument-1}) ** (\text{argument-2}))$$

## ASIN

The ASIN function returns a numeric value, expressed in radians, that approximates the arcsine of argument -1. The type of this function is floating point.

### *Format*

FUNCTION ASIN(argument-1)

### *Arguments*

1. Argument-1 must be class numeric.
2. The value of argument-1 must be greater than or equal to -1 and less than or equal to + 1.

### *Returned Value*

The returned value is the approximation of the arcsine of argument-1 and is greater than or equal to -pi/2 and less than or equal to + pi/2.

## ATAN

The ATAN function returns a numeric value, expressed in radians, that approximates the arctangent of argument-1. The type of this function is floating point.

### *Format*

FUNCTION gTAN(argument-1)

### *Argument*

Argument-1 must be class numeric.

### *Returned Value*

The returned value is the approximation of the arctangent of argument- 1 and is greater than -pi/2 and less than + pi/2.

## CHAR

The CHAR function returns a 1-character alphanumeric value that is a character in the program collating sequence having the ordinal position equal to the value of argument-1. The type of this function is alphanumeric.

### Format

FUNCTION CHAR (argument-1)

### Arguments

1. Argument-1 must be an integer.
2. The value of argument-1 must be greater than zero and less than or equal to the number of positions in the collating sequence.

### *Returned Values*

1. If more than one character has the same position in the program collating sequence, the character returned as the function value is that of the first literal specified for that character position in the ALPHABET clause.
2. If the current program collating sequence was not specified by an ALPHABET clause, the implementor determines the value.

## COS

The COS function returns a numeric value that approximates the cosine of an angle or arc, expressed in radians, that is specified by argument-1. The type of this function is floating point.

### Format

FUNCTION COS (argument-1)

### Argument

Argument-1 must be class numeric.

### *Returned Value*

The returned value is the approximation of the cosine of argument-1 and is greater than or equal to -1 and less than or equal to +1.

## CURRENT-DATE

The CURRENT-DATE function returns a 21-character alphanumeric value that represents the calendar date, time of day, and local time differential factor provided by the system on which the function is evaluated. The type of this function is alphanumeric.

### **Format**

FUNCTION CURRENT-DATE

### **Returned Values**

The character positions returned, numbered from left to right, are shown in the following list:

<b>Character Positions</b>	<b>Contents</b>
1-4	Four numeric digits of the year in the Gregorian calendar.
5-6	Two numeric digits of the month of the year, in the range 01 through 12.
7-8	Two numeric digits of the day of the month, in the range 01 through 31.
9-10	Two numeric digits of the hours past midnight, in the range 00 through 23.
11-12	Two numeric digits of the minutes past the hour, in the range 00 through 59.
13-14	Two numeric digits of the seconds past the minute, in the range 00 through 59.
15-16	Two numeric digits of the hundredths of a second past the second, in the range 00 through 99. The value 00 is returned if the system on which the function is evaluated does not have the facility to provide the fractional part of a second.
17	The character '-', the character '+', or the character '0'. The character '-' is returned if the local time indicated in the previous character positions is behind Greenwich mean time. The character '+' is returned if the local time indicated is the same as or ahead of Greenwich mean time. The character '0' is returned if the system on which this function is evaluated does not have the facility to provide the local time differential factor.
18-19	If character position 17 is '-', two numeric digits are returned in the range 00 through 12 indicating the number of hours that the reported time is behind Greenwich mean time. If character position 17 is '+', two numeric digits are returned in the range 00 through 13 indicating the number of hours that the reported time is ahead of Greenwich mean time. If character position 17 is '0', the value 00 is returned.
20-21	Two numeric digits are returned in the range 00 through 59 indicating the number of additional minutes that the reported time is ahead of or behind Greenwich mean time, depending on whether character position 17 is '+' or '-', respectively. If character position 17 is '0', the value 00 is returned.

## DATE-OF-INTEGER

The DATE-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to standard date form (YYYYMMDD). The type of this function is integer.

### *Format*

FUNCTION 'DATE-OF-INTEGER' (argument-1)

### *Argument*

Argument 1 is a positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar.

### *Returned Values*

1. The returned value represents the ISO Standard date equivalent of the integer specified in argument-1.
2. The returned value is in the form (YYYYMMDD) where YYYY represents a year in the Gregorian calendar, MM represents the month of that year, and DD represents the day of that month.

## DAY-OF-INTEGER

The DAY-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to Julian date form (YYYYDDD). The type of this function is integer.

### *Format*

FUNCTION DAY-OF-INTEGER' (argument-1)

### *Argument*

Argument- I is a positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar.

### *Returned Values*

1. The returned value represents the Julian equivalent of the integer specified in argument -1.
2. The returned value is an integer of the form (YYYYDDD) where YYYY represents a year in the Gregorian calendar, and DDD represents the day of that year.

## FACTORIAL

The FACTORIAL function returns an integer that is the factorial of argument-1. The type of this function is integer.

### *Format*

FUNCTION FACTORIAL (argument-1)

### *Argument*

Argument-1 must be an integer greater than or equal to zero and less than 29.

### *Returned Values*

1. If the value of argument-1 is zero, the value 1 is returned
2. If the value of argument-1 is positive, its factorial is returned.

## INTEGER

The INTEGER function returns the greatest integer value that is less than or equal to the argument. The type of this function is integer.

### *Format*

FUNCTION INTEGER (argument-1)

### *Argument*

Argument-1 must be class numeric.

### *Returned Value*

The returned value is the greatest integer less than or equal to the value of argument-1. For example, if the value of argument-1 is -1.5, -2 is returned. If the value of argument-1 is +1.5, +1 is returned.

## INTEGER-OF-DATE

The INTEGER-OF-DATE function converts a date in the Gregorian calendar from standard date form (YYYYMMDD) to integer date form. The type of this function is integer.

### *Format*

FUNCTION INTEGER-OF-DATE (argument-1)

### *Argument*

Argument-1 must be an integer of the form YYYYMMDD, whose value is obtained from the calculation  $(YYYY * 10,000) + (MM * 100) + DD$ . The following rules apply:

- YYYY represents the year in the Gregorian calendar. It must be an integer greater than 1600.
- MM represents a month and must be a positive integer less than 13.
- DD represents a day and must be a positive integer less than 32, provided that it is valid for the specified month and year combination.

### *Returned Value*

The returned value is an integer that is the number of days the date represented by argument-1 succeeds December 31, 1600, in the Gregorian calendar.

## INTEGER-OF-DAY

The INTEGER-OF-DAY function converts a date in the Gregorian calendar from Julian date form (YYYYDDD) to integer date form. The type of this function is integer.

### *Format*

FUNCTION INTEGER-OF-DAY (argument-1)

### *Argument*

Argument-1 must be an integer of the form YYYYDDD, whose value is obtained from the calculation  $(YYYY * 1000) + DDD$ .

- YYYY represents the year in the Gregorian calendar. It must be an integer greater than 1600.
- DDD represents the day of the year. It must be a positive integer less than 367, provided that it is valid for the year specified.

### *Returned Value*

The returned value is an integer that is the number of days the date represented by argument-1 succeeds December 31, 1600, in the Gregorian calendar.



## INTEGER-PART

The INTEGER-PART function returns an integer that is the integer portion of argument-1. The type of this function is integer.

### Format

FUNCTION INTEGER-PART                      (argument-1)

### Argument

Argument-1 must be class numeric.

### *Returned Values*

1. If the value of argument-1 is zero, the returned value is zero.
2. If the value of argument-1 is positive, the returned value is the greatest integer less than or equal to the value of argument-1. For example, if the value of argument-1 is + 1.5, +1 is returned.
3. If the value of argument-1 is negative, the returned value is the least integer greater than or equal to the value of argument-1. For example, if the value of argument-1 is -1.5, -1 is returned.

## LENGTH

The LENGTH function returns an integer equal to the length of the argument in character positions. The type of this function is integer.

### Format

FUNCTION LENGTH (argument-1)

### Arguments

1. Argument-1 can be a nonnumeric literal or a data item of any class or category.
2. If argument-1 or any data item subordinate to argument-1 is described with the DEPENDING phrase of the OCCURS clause, the contents of the data item referenced by the data-name specified in the DEPENDING phrase are used at the time the LENGTH function is evaluated

### Returned Values

1. If argument-1 is a nonnumeric literal or an elementary data item, or argument-1 is a group data item that does not contain a variable occurrence data item, the value returned is an integer equal to the length of argument-1 in character positions.
2. If argument-1 is a group data item containing a variable occurrence data item, the returned value is an integer determined by evaluation of the data item specified in the DEPENDING phrase of the OCCURS clause for that variable occurrence data item. This evaluation is accomplished according to the rules in the OCCURS clause dealing with the data item as a sending data item.
3. The returned value includes implicit FILLER characters, if any.

## LOG

The LOG function returns a numeric value that approximates the logarithm to the base a (natural log) of argument-1. The type of this function is floating point.

### *Format*

FUNCTION LOG (argument-1)

### *Arguments*

1. Argument-1 must be class numeric.
2. The value of argument-1 must be greater than zero.

### *Returned Value*

The returned value is the approximation of the logarithm to the base a of argument-1.

## LOG10

The LOG 10 function returns a numeric value that approximates the logarithm to the base 10 of argument-1. The type of this function is floating point.

### *Format*

FUNCTION LOG10 (argument -1)

### *Arguments*

1. Argument-1 must be class numeric.
2. The value of argument-1 must be greater than zero.

### *Returned Value*

The returned value is the approximation of the logarithm to the base 10 of argument-1.

## LOWER-CASE

The LOWER-CASE function returns a character string that is the same length as argument-1 with each uppercase letter replaced by the corresponding lowercase letter. The type of this function is alphanumeric.

### *Format*

FUNCTION LOWER-CASE(argument-1)

### *Argument*

Argument-1 must be class alphabetic or alphanumeric and must be at least one character in length.

### *Returned Values*

1. The same character string as argument-1 is returned, except that each uppercase letter is replaced by the corresponding lowercase letter.
2. The character string returned has the same length as argument-1.
3. If the computer character set does not include lowercase letters, no changes take place in the character string.

## MAX

The MAX function returns the content of the argument-1 that contains the maximum value. The type of this function depends upon the argument types as follows:

<b><u>Argument Type</u></b>	<b><u>Function Type</u></b>
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
All arguments integer	Integer
Numeric (some arguments can be integer)	Numeric

### Format

**FUNCTION MAX** ((argument-1) ... )

### Arguments

If more than one argument-1 is specified, all arguments must be of the same class, except that mixing of arguments of alphabetic and alphanumeric classes is allowed.

### Returned Values

1. The returned value is the content of the argument-1 having the greatest value. The comparisons used to determine the greatest value are made according to the rules for simple conditions.
2. If more than one argument-1 has the same greatest value, the content of the argument-1 returned is the leftmost argument-1 having that value.
3. If the type of the function is alphanumeric, the size of the returned value is the same as the size of the selected argument-1.

## MEAN

The MEAN function returns a numeric value that is the arithmetic mean (average) of its arguments. The type of this function is numeric.

### Format

FUNCTION MEAN((argument-1) ... )

### Argument

Argument-1 must be class numeric.

### Returned Values

1. The returned value is the arithmetic mean of the argument-1 series.
2. The returned value is defined as the sum of the argument-1 series divided by the number of occurrences referenced by argument-1.

## MEDIAN

The MEDIAN function returns the content of the argument whose value is the middle value in the list formed by arranging the arguments in sorted order. The type of this function is numeric.

### Format

FUNCTION MEDIAN( { argument-1 } ... )

### Argument

Argument-1 must be class numeric.

### Returned Values

1. The returned value is the content of the argument-1 having the middle value in the list formed by arranging all the argument-1 values in sorted order.
2. If the number of occurrences referenced by argument-1 is odd, *the returned* value is such that at least half of the occurrences referenced by argument-1 are greater than or equal to the returned value and at least half are less than or equal. If the number of occurrences referenced by argument-1 is even, the returned value is the arithmetic mean of the values referenced by the two middle occurrences.
3. The comparisons used to arrange the argument-1 values in sorted order are made according to the rules for simple conditions.

## MIDRANGE

The MIDRANGE (middle range) function returns a numeric value that is the arithmetic mean (average) of the values of the minimum argument and the maximum argument. The type of this function is numeric.

### *Format*

FUNCTION MIDRANGE((argument-1) ...)

### *Argument*

Argument-1 must be class numeric.

### *Returned Value*

The returned value is the arithmetic mean of the greatest argument-1 value and the least argument-1 value. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions.

## MIN

The MIN function returns the content of the argument-1 that contains the minimum value. The type of this function depends upon the argument types as follows:

<u>Argument Type</u>	<u>Function Type</u>
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
All arguments integer	Integer
Numeric (some arguments can be integer)	Numeric

### *Format*

FUNCTION MIN((argument-1) ...)

### *Argument*

If more than one argument-1 is specified, all arguments must be of the same class, except that mixing of arguments of alphabetic and alphanumeric classes is allowed.

### *Returned Values*

1. The returned value is the content of the argument-1 having the least value. The comparisons used to determine the least value are made according to the rules for simple conditions.
2. If more than one argument-1 has the same least value, the content of the argument-1 returned is the leftmost argument-1 having that value.
3. If the type of the function is alphanumeric, the size of the returned value is the same as the size of the selected argument-1.

## MOD

The MOD (modules) function returns an integer value that is argument-1 modulo argument-2. The type of this function is integer.

### Format

FUNCTION MOD (argument-1 argument-2)

### Arguments

1. Argument-1 and argument-2 must be integers.
2. The value of argument-2 must not be zero.

### Returned Values

1. The returned value is argument-1 modulo argument-2. The returned value is defined as  
$$\text{argument-1} - (\text{argument-2} * \text{FUNCTION INTEGER}(\text{argument-1}/\text{argument-2}))$$
2. The following illustrates the expected results for some values of argument-1 and argument-2.

Argument-1	Argument-2	Return
11	5	1
-11	5	4
11	-5	-4
-11	-5	-1



## NUMVAL

The NUMVAL function returns the numeric value represented by the character string specified by argument-1. Leading and trailing spaces are ignored. The type of this function is floating point.

### Format

FUNCTION NUMVAL (argument-1)

### Argument

1. Argument-1 must be a nonnumeric literal or alphanumeric data item whose content has one of the following two formats:

$$[\text{space}] \left[ \begin{array}{c} + \\ - \end{array} \right] [\text{space}] \left\{ \begin{array}{l} \text{digit} [ . [\text{digit}] ] \\ . \text{digit} \end{array} \right\} [\text{space}]$$

or

$$[\text{space}] \left\{ \begin{array}{l} \text{digit} [ . [\text{digit}] ] \\ . \text{digit} \end{array} \right\} [\text{space}] \left[ \begin{array}{c} + \\ - \\ \text{CR} \\ \text{DB} \end{array} \right] [\text{space}]$$

where space is a string of zero or more spaces, and digit is a string of 1 to 18 digits.

2. The total number of digits in argument-1 must not exceed 18.
3. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIALNAMES paragraph, a comma must be used in argument-1 rather than a DECIMAL point.

### Returned Values

1. The returned value is the numeric value represented by argument-1.
2. The number of digits returned is 18.

## NUMVAL-C

The NUMVAL-C function returns the numeric value represented by the character string specified by argument-1. Any optional currency sign specified by argument-2 and any optional commas preceding the decimal point are ignored. The type of this function is floating point.

### Format

FUNCTION NUMVAL-C (argument-1 [argument-2] )

### Arguments

1. Argument-1 must be a nonnumeric literal or alphanumeric data item whose content has one of the following two formats:

$$[\text{space}] \begin{bmatrix} + \\ - \end{bmatrix} [\text{space}] [\text{cs}] [\text{space}] \left\{ \begin{array}{l} \text{digit} [ . \text{digit} ] \dots [ . \text{digit} ] \\ . \text{digit} \end{array} \right\} [\text{space}]$$

or

$$[\text{space}] [\text{cs}] [\text{space}] \left\{ \begin{array}{l} \text{digit} [ . \text{digit} ] \dots [ . \text{digit} ] \\ . \text{digit} \end{array} \right\} [\text{space}] \begin{bmatrix} + \\ - \\ \text{CR} \\ \text{DB} \end{bmatrix} [\text{space}]$$

where space is a string of zero or more spaces, cs is the string of one or more characters specified by argument-2, and digit is a string of one or more digits.

2. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIALNAMES paragraph, the functions of the comma and decimal point in argument-1 are reversed.
3. The total number of digits in argument-1 must not exceed 18.
4. Argument-2, if specified, must be a nonnumeric literal or alphanumeric data item.
5. If argument-2 is not specified, the character used for cs is the currency symbol specified for the program.

### Returned Values

1. The returned value is the numeric value represented by argument-1.
2. The number of digits returned is 18.

## ORD

The ORD function returns an integer value that is the ordinal position of argument-1 in the collating sequence for the program. The lowest ordinal position is 1. The type of this function is integer.

### *Format*

FUNCTION ORD(argument-1)

### *Argument*

Argument-1 must be one character in length and must be class alphabetic or alphanumeric.

### *Returned Value*

The returned value is the ordinal position of argument-1 in the collating sequence for the program.

## ORD-MAX

The ORD-MAX function returns a value that is the ordinal number of the argument-1 that contains the maximum value. The type of this function is integer.

### *Format*

FUNCTION ORD-MAX      ((argument-1) ... )

### *Arguments*

If more than one argument-1 is specified, all arguments must be of the same class, except that mixing arguments of alphabetic and alphanumeric classes is allowed

### *Returned Values*

1. The returned value is the ordinal number that corresponds to the position of the argument-1 having the greatest value in the argument-1 series.
2. The comparisons used to determine the greatest valued argument are made according to the rules for simple conditions.
3. If more than one argument-1 has the same greatest value, the number returned corresponds to the position of the leftmost argument-1 having that value.

## ORD-MIN

The ORD-MIN function returns a value that is the ordinal number of the argument that contains the minimum value. The type of this function is integer.

### Format

FUNCTION ORD-MIN ((argument-1? ... )

### Arguments

If more than one argument-1 is specified, all arguments must be of the same class, except that mixing arguments of alphabetic and alphanumeric classes is allowed.

### Returned Values

1. The returned value is the ordinal number that corresponds to the position of the argument-1 having the least value in the argument-1 series.
2. The comparisons used to determine the least valued argument-1 are made according to the rules for simple conditions.
3. If more than one argument-1 has the same least value, the number returned corresponds to the position of the leftmost argument-1 having that value.

## PRESENT-VALUE

The PRESENT-VALUE function returns a value that approximates the present value of a series of future period-end amounts specified by argument-2 at a discount rate specified by argument-1. The type of this function is floating point.

### Format

FUNCTION PRESENT-VALUE (argument-1 (argument-2) ... )

### Arguments

1. Argument-1 and argument-2 must be of the class numeric.
2. The value of argument-1 must be greater than -1.

### Returned Value

The returned value is an approximation of the summation of a series of calculations with each term in the following form:

$$\text{argument-2} / (1 + \text{argument-1}^{**} n)$$

There is one term for each occurrence of argument-2. The exponent, n, is incremented from 1, by one for each term in the series.

## **RANDOM**

The RANDOM function returns a numeric value that is a pseudo-random number from a rectangular distribution. The type of this function is numeric.

### ***Format***

FUNCTION RANDOM [ ( argument-1 ) ]

### ***Argument***

1. If argument-1 is specified, it must be zero or a positive integer. It is used as the seed value to generate a sequence of pseudo-random numbers.
2. If a subsequent reference specifies argument-1, a new sequence of pseudorandom numbers is started.
3. If the first reference to this function in the run unit does not specify argument-1, the seed value is defined by the implementor.
4. In each case, subsequent references without specifying argument-1 return the next number in the current sequence.

### ***Returned Values***

1. The returned value is greater than or equal to zero and less than one.
2. For a given seed value on a given implementation, the sequence of pseudorandom numbers will always be the same.
3. The subset of the domain of argument-1 values that will yield distinct sequences of pseudo-random numbers includes the values 0 through 32,767.

## RANGE

The RANGE function returns a value that is equal to the value of the maximum argument minus the value of the minimum argument. The type of this function depends upon the argument types as follows:

<u>Argument Type</u>	<u>Function Type</u>
All arguments integer	Integer
Numeric (some arguments can be integer)	Numeric

### *Format*

FUNCTION RANGE ((argument-1) . . . )

### *Argument*

Argument-1 must be class numeric.

### *Returned Values*

The returned value is equal to the greatest value of argument-1 minus the least value of argument-1. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions.

## REM

The REM function returns a numeric value that is the remainder of argument-1 divided by argument-2. The type of this function is numeric.

### *Format*

FUNCTION \$M (argument-1 argument-2)

### *Arguments*

1. Argument-1 and argument-2 must be class numeric.
2. The value of argument-2 must not be zero.

### *Returned Value*

The returned value is the remainder of argument-1 / argument-2. It is defined as the expression  
$$\text{argument-1} - (\text{argument-2} * \text{FUNCTION INTEGER-PART} (\text{argument-1} / \text{argument-2}))$$

## REVERSE

The REVERSE function returns a character string of exactly the same length as argument-1 and whose characters are exactly the same as those of argument-1, except that they are in reverse order. The type of this function is alphanumeric.

### *Format*

FUNCTION REVERSE (argument-1)

### *Argument*

Argument-1 must be class alphabetic or alphanumeric and must be at least one character in length.

### *Returned Value*

If argument-1 is a character string of length  $n$ , the returned value is a character string of length  $n$  such that for  $1 \leq j \leq n$ , the character in position  $j$  of the returned value is the character from position  $n - j + 1$  of argument-1.

## SIN

The SIN function returns a numeric value that approximates the sine of an angle or arc, expressed in radians, that is specified by argument-1. The type of this function is floating point.

### *Format*

FUNCTION SIN (argument-1)

### *Argument*

Argument-1 must be class numeric.

### *Returned Value*

The returned value is the approximation of the sine of argument-1 and is greater than or equal to - 1 and less than or equal to +1.

## SQRT

The SQRT function returns a numeric value that approximates the square root of argument-1. The type of this function is floating point.

### *Format*

FUNCTION SQRT (argument-1)

### *Argument*

1. Argument-1 must be class numeric. 2. The value of argument-1 must be zero or positive.

### *Returned Value*

The returned value is the absolute value of the approximation of the square root of argument-1.

## STANDARD-DEVIATION

The STANDARD-DEVIATION function returns a numeric value that approximates the standard deviation of its arguments. The type of this function is numeric.

### *Format*

FUNCTION STANDARD-DEVIATION ((argument-1) ...)

### *Argument*

Argument-1 must be class numeric.

### *Returned Values*

1. The returned value is the approximation of the standard deviation of the argument-1 series.
2. The returned value is calculated as follows:
  - a. The difference between each argument-1 value and the arithmetic mean of the argument-1 series is calculated and squared
  - b. The values obtained are then added together. This quantity is divided by the number of values in the argument-1 series.
  - c. The square root of the quotient obtained is then calculated. The returned value is the absolute value of this square root.
3. If the argument-1 series consists of only one value, or if the argument-1 series consists of all variable occurrence data items and the total number of occurrences for all of them is one, the returned value is zero.



## SUM

The SUM function returns a value that is the sum of the arguments. The type of this function depends upon the argument types as follows:

<u>Argument Type</u>	<u>Function Type</u>
All arguments integer	Integer
Numeric (some arguments can be integer)	Numeric

### *Format*

FUNCTION SUM ((argument-1) . . . )

### *Argument*

Argument-1 must be class numeric.

### *Returned Values*

1. The returned value is the sum of the arguments.
2. If the argument-1 series are all integers, the value returned is an integer.
3. If the argument-1 series are not all integers, a numeric value is returned.

## TAN

The TAN function returns a numeric value that approximates the tangent of an angle or arc, expressed in radians, that is specified by argument-1. The type of this function is floating point.

### *Format*

FUNCTION TAN (argument-1)

### *Argument*

Argument-1 must be class numeric.

### *Returned Value*

The returned value is the approximation of the tangent of argument-1.

## UPPER-CASE

The UPPER-CASE function returns a character string that is the same length as argument-1 with each lowercase letter replaced by the corresponding uppercase letter. The type of this function is alphanumeric.

### *Format*

FUNCTION UPPER-CASE(argument-1)

### *Argument*

Argument-1 must be class alphabetic or alphanumeric and must be at least one character in length.

### *Returned Values*

1. The same character string as argument-1 is returned except that each lowercase letter is replaced by the corresponding uppercase letter.
2. The character string returned has the same length as argument-1.

## VARIANCE

The VARIANCE function returns a numeric value that approximates the variance of its arguments. The type of this function is numeric.

### *Format*

FUNCTION VARIANCE((argument-1) ... )

### *Argument*

Argument-1 must be class numeric.

### *Returned Values*

1. The returned value is the approximation of the variance of the argument-1 series.
2. The returned value is defined as the square of the standard deviation of the argument-1 series. Refer to the section titled "STANDARD-DEVIATION," Returned Values, rule 2.
3. If the argument-1 series consists of only one value, or if the argument- I series consists of all variable occurrence data items and the total number of occurrences for all of them is one, the returned value is zero.

## WHEN-COMPILED

The WHEN-COMPILED function returns the date and time the program was compiled as provided by the system on which the program was compiled. The type of this function is alphanumeric.

### *Format*

FUNCTION WHEN-COMPILED

### *Returned Values*

1. The character positions returned, numbered from left to right, are shown in the following list:

<u>Character Positions</u>	<u>Contents</u>
--------------------------------	-----------------

- |       |  |
|-------|--|
| 1-4   | Four numeric digits of the year in the Gregorian calendar.   |
| 5-6   | Two numeric digits of the month of the year, in the range 01 through 12.   |
| 7-8   | Two numeric digits of the day of the month, in the range 01 through 31.  |
| 9-10  | Two numeric digits of the hours past midnight, in the range 00 through 23.   |
| 11-12 | Two numeric digits of the minutes past the hour, in the range 00 through 59.   |
| 13-14 | Two numeric digits of the seconds past the minute, in the range 00 through 59.   |
| 15-16 | Two numeric digits of the hundredths of a second past the second, in the range 00 through 99. The value 00 is returned if the system on which the function is evaluated does not have the facility to provide the fractional part of a second.   |
| P17   | The character '-', the character '+', or the character '0'. The character '-' is returned if the local time indicated in the previous character positions is behind Greenwich mean time. The character '+' is returned if the local time indicated is the same as or ahead of Greenwich mean time. The character '0' is returned if the system on which this function is evaluated does not have the facility to provide the local time differential factor. |
| 18-19 | If character position 17 is '-', two numeric digits are returned in the range 00 through 12 indicating the number of hours that the reported time is behind Greenwich mean time. If character position 17 is '+', two numeric digits are returned in the range 00 through 13 indicating the number of hours that the reported time is ahead of Greenwich mean time. If character position 17 is '0', the value 00 is returned.                               |
| 20-21 | Two numeric digits are returned in the range 00 through 59 indicating the number of additional minutes that the reported time is ahead of or behind Greenwich mean time, depending on whether character position 17 is '+' or '-', respectively. If character position 17 is '0', the value 00 is returned.  |
2. The returned value is the date and time of compilation of the source program that contains this function. The returned value is the compilation date and time associated with the separately compiled program in which it is contained.
  3. The returned value denotes the same time as the compilation date and time provided in the listing of the source program and in the generated object code for the source program, although their representations and precisions can differ.

# Appendix A Reserved Words

## Introduction

This appendix lists both COBOL reserved words and words that are reserved by the Wang implementation of COBOL.

## COBOL Reserved Words

The following words are reserved by the COBOL Standard.

ACCEPT	ANY	CALL
ACCESS	ARE	CANCEL
ADD	AREA	CD
ADVANCING	AREAS	CF
AFTER	ASCENDING	CH
ALL	ASSIGN	CHARACTER
ALPHABET	AT	CHARACTERS
ALPHABETIC	AUTHOR	CLASS
ALPHABETIC-LOWER		CLOCK-UNITS
ALPHABETIC-UPPER	BEFORE	CLOSE
ALPHANUMERIC	BINARY	COBOL
ALPHANUMERIC-EDITED	BLANK	CODE
ALSO	BLOCK	CODE-SET
ALTER	BOTTOM	COLLATING
ALTERNATE BY	COLUMN	
AND		
COMMA	DELIMITER	ENTER
COMMON	DEPENDING	ENVIRONMENT
COMMUNICATION	DESCENDING	EOP
COMP	DESTINATION	EQUAL
COMPUTATIONAL	DETAIL	ERROR
COMPUTE	DISABLE	ESI
CONFIGURATION	DISPLAY	EVALUATE
CONTAINS	DIVIDE	EVERY
CONTENT	DIVISION	EXCEPTION
CONTINUE	DOWN	EXIT
CONTROL	DUPLICATE	EXTEND
CONTROLS	DYNAMIC	EXTERNAL
CONVERTING		
COPY	EGI	FALSE
CORR	ELSE	FD
CORRESPONDING	EMI	FILE
COUNT	ENABLE	FILE-CONTROL

CURRENCY	END	FILLER
DATA	END-ADD	FINAL
DATE	END-CALL	FIRST
DATE-COMPILED	END-COMPUTE	FOOTING
DATE-WRITTEN	END-DELETE	FOR
DAY	END-DIVIDE	FROM
DAY-OF-WEEK	END-EVALUATE	FUNCTION
DE	END-IF	
DEBUG-CONTENTS	END-MULTIPLY	GENERATE
DEBUG-ITEM	END-OF-PAGE	GIVING
DEBUG-LINE	END-PERFORM	GLOBAL
DEBUG-NAME	END-READ	GO
DEBUG-SUB- 1	END-RECEIVE	GREATER
DEBUG-SUB-2	END-RETURN	GROUP
DEBUG-SUB-3	END-REWRITE	
DEBUGGING	END-SEARCH	HEADING
DECIMAL-POINT	END-START	HIGH-VALUE
DECLARATIVES	END-STRING	HIGH-VALUES
DELETE	END-SUBTRACT	
DELIMITED	END-UNSTRING	I-O
	END-WRITE	I-O-CONTROL
IDENTIFICATION	LOW-VALUES	PAGE-COUNTER
IF		PERFORM
IN	MEMORY	PF
INDEX	MERGE	PH
INDEXED	MESSAGE	PIC
INDICATE	MODE	PICTURE
INITIAL	MODULES	PLUS
INITIALIZE	MOVE	POINTER
INITIATE	MULTIPLE	POSITION
INPUT	MULTIPLY	POSITIVE
INPUT-OUTPUT	PRINTING	
INSPECT	NATIVE	PROCEDURE
INSTALLATION	NEGATIVE	PROCEDURES
INTO	NEXT	PROCEED
INVALID	NO	PROGRAM
IS	NOT	PROGRAM-ID
	NUMBER	PURGE
JUST	NUMERIC	
JUSTIFIED	NUMERIC-EDITEP	QUEUE
		QUOTE
KEY	OBJECT-COMPUTER	QUOTES
	OCCURS	
LABEL	OF	RANDOM
LAST	OFF	RD
LEADING	OMITTED	READ
LEFT	ON	RECEIVE
LENGTH	OPEN	RECORD
LESS	OPTIONAL	RECORDS
LIMIT	OR	REDEFINES

LIMITS	ORDER	REEL
LINAGE	ORGANIZATION	REFERENCE
LINAGE-COUNTER	OTHER	REFERENCES
LINE	OUTPUT	RELATIVE
LINE-COUNTER	OVERFLOW	RELEASE
LINES	REMAINDER	
LINKAGE	PACKED-DECIMAL	REMOVAL
LOCK PADDING		RENAMES
LOW-VALUEPAGE		
REPLACE	SOURCE-COMPUTER TYPE	
REPLACING	SPACE	
REPORT	SPACES	UNIT
REPORTING	SPECIAL-NAMES	UNSTRING
REPORTS	STANDARD	UNTIL
RERUN	STANDARD-1	UP
RESERVE	STANDARD-2	UPON
RESET	START	USAGE
RETURN	STATUS	USE
RETURN-CODE	STOP	USING
REVERSED	STRING	
REWIND	SUB-QUEUE-1	VALUE
REWRITE	SUB-QUEUE-2	VALUES
RF	SUB-QUEUE-3	VARYING
RH	SUBTRACT	
RIGHT	SUM	WHEN
ROUNDED	SUPPRESS	WITH
RUN	SYMBOLIC	WORDS
	SYNC	WORKING-STORAGE
SAME	SYNCHRONIZED	WRITE
SD		
SEARCH	TABLE	ZERO
SECTION	TALLYING	ZEROES
SECURITY	TAPE.	ZEROS
SEGMENT	TERMINAL	
SEGMENT-LIMIT	TERMINATE	+
SELECT	TEST	-
SEND	TEXT	*
SENTENCE	THAN	/
SEPARATE	THEN	**
SEQUENCE	THROUGH	>
SEQUENTIAL	THRU	<
SET	TIME	=
SIGN	TIMES	> =
SIZE	TO	< =
SORT	TOP	
SORT-MERGE	TRAILING	
SOURCE	TRUE	

## Words Reserved by Wang

The following words are reserved by the Wang implementations of COBOL. The words relate to Interactive processing, File Sharing, Transaction control, and other Wang extensions to the Standard.

ALARM	HASHSIZE	ROLL
ALTERED	HOLD	ROLLBACK
	HOLDER-ID	ROW
BEGIN		
BLOCKS	INVOKE	SECOND
BUFFER		SECONDS
	KEYS	SETTING
COMMIT		SHARED
COMPRESSED	LIST	SPECIAL-INPUT
CONVERSION		SUBTRANSACTION
CURSOR	MODIFIABLE	
	MODIFY	TIMEOUT
DEADLOCK		TRACE
DELETION	NO-MOD	TRANSACTION
DISPLAY-WS	NODISPLAY	
	NORESPECIFY	UPDATE
END-BEGIN		
END-COMMIT	OBJECT	
END-DISPLAY	ONLY	
END-FREE	ORDER-AREA	
END-HOLD		
END-MOVE	PFKEY	
END-ROLLBACK	PFKEYS	
ERASE	PRIOR	
EXCLUSIVE	PROTECT	
EXTENSION-RIGHTS		
	RANGE	
FAC	READY	
FIGURATIVE-	RESPECIFY	
CONSTANTS	RESTART	
FREE	RETRIEVAL	

# Appendix B Compiler Options and Directives

## Introduction

This appendix describes the Compiler options and source text directives that affect compiler inputs and outputs and object code performance.

## Changing Compiler Options Defaults on the VS

The Compiler option defaults on the VS can be changed using SETOPTS, a standalone utility that enables users to specify installation default values for programs that use Forms files. These defaults appear whenever the Options screen is displayed by the compiler or by a program that invokes the compiler.

When SETOPTS is run, it displays the Program Selection screen, which displays a list of available tool and utility names from the Forms Index file: @FORMS@ in @SYSFRM@. In order to change the COBOL 85 options defaults, the users position the cursor at COBOL85 or enter COBOL85 into the PROGRAM field of the screen, then press RETURN.

The system then displays the Screen Selection screen, in which the users select the screen containing the options to be changed. The users position the cursor at OPTIONS, or enter OPTIONS in the SCREEN field, and press RETURN. The Compiler Options screen is then displayed, and the users can alter the option defaults.

To change the defaults in a screen other than the Compiler Options screen, users must first set the option in the Compiler Options screen that triggers the screen whose defaults are to be altered. For instance, if users want to set defaults in the LINKLIBS screen, they set the MORE field to YES. The LINKLIBS screen appears.

The SETOPTS utility can be run interactively or from a procedure. SETOPTS can be run from a procedure that provides values for both SETOPTS screens and the COBOL 85 Options screens. The SETOPTS pmames and keywords are SELPROG - PROGRAM and SELSCRN - SCREEN. The user is required to know the keywords for the default values to be modified. When a screen is displayed as the result of a procedural "DISPLAY", values specified in the procedure override the Forms file values. If any default values are in error, the screen is displayed, even though the procedure used "ENTER".



## Compiler Options

The Compiler options are organized on the options prompt screen according to frequency of use and type. The descriptions of the options are presented here in alphabetical order by keyword.

Options that are not supported by COBOL ReSource are indicated by the words "Not supported" in the text of the *Command Line*. The word "Omit" for a *Command Line* value indicates that option value above it is the *COBOL* ReSource default assumed for this option.

### Default Buffer Pool Blocks

Keyword: BUFPOOL            Value: n

Command Line:              Not supported

The parameter n must be from 3 to 256. It specifies the number of buffer blocks for indexed files. This option is the equivalent of the following COBOL 85 statements:

```
RESERVE n AREAS
```

and

```
SAME AREA index-file1, index-file2
```

### Default Buffer Size in Blocks

Keyword: BUFSIZE            Value: n

Command Line:              Not supported

The parameter n must be from 1 to 9. It specifies the number of blocks for sequential and relative files. This option is the equivalent of the following COBOL 85 statements:

```
BUFFER SIZE IS n BLOCKS
```

or

```
RESERVE n AREAS
```

### Allow CANCEL verb

Keyword: CANCEL            Values: YES            NO

Command Line:              -q            Omit

If CANCEL = YES, programs in this compilation unit can be referenced in a CANCEL statement. If CANCEL = NO, any attempt to cancel any one of the programs in this compilation unit will result in a program check at runtime.

### COBOL 74 Compatibility Level

Keyword: COMPAT74           Values: YES            IO        NUC    NO

Command Line:                -v1        -v2        -v3       -v0

With COMPAT74=NO, COBOL 85 rules are enforced. With COMPAT74=IO, 10 statement semantics follow the COBOL 74 rules. With COMPAT--NUC, the size of fields defined as USAGE INDEX with no PICTURE clause, certain comparison semantics, and certain MOVE semantics follow the COBOL 74 rules. Specifying COMPAT74=YES is equivalent to specifying COMPAT74=IO and COMPAT74=NUC simultaneously. The default is 10.

The goal of this option is to provide compatibility with and support to gradual migration from the COBOL 74 to the COBOL 85 Standard. For details, refer to the *COBOL 85 Migration Guide*, July 1992.

## COPY Libraries

Keyword: COPYLIBS      Values: YES      NO

Command Line: `-I pathname` Omit

When a COPY statement is encountered that does not provide the full pathname of the text, the COPYLIBS option allows up to 16 pathnames to be specified as the libraries or volumes to be searched.

If COPYLIBS = YES, the compiler prompts with the COPYLIBS screen to obtain up to eight volume and library names to be used in the COPY file search. To specify more than eight copy libraries, the user must set the MORE keyword on the COPYLIBS screen to YES. This causes display of the ADDLIBS screen, which allows the user to specify up to eight additional libraries.

On a UNIX system, the Command Line option `I pathname` implicitly sets `COPYLIB` to `YES` as it provides the additional pathname to search. Up to 16 repetitions of this option are supported.

If COPYLIBS = NO, the source library is used.

The specified libraries are searched in the order in which they appear, so attention has to be paid to files with identical names. When a `COPY` statement specifies only a filename, only those `COPYLIBS` entries that specify both a library name and volume name are used in the search. (Entries containing only a volume name are ignored.)

When a COPY statement specifies a file and library name, but no volume, only those COPYLIBS entries that contain just a volume name are used in the search. (Entries that contain both a library and volume name are ignored.) The last volume searched is the default INVOL.

## Produce a Data Division Map

Keyword: DMAP      Values: YES   NO

Command Line:                -d      Omit

If DMAP = YES, a Data Division Map is produced. The elements of the map are

**Name** - The name of the data-item. If FILLER or no name was given, "FILLER" is printed as the name. If the data-item is a member of a group, the name is indented and preceded by a number that shows the nesting level.

**Offset**-The displacement in bytes from the start of this static area. It is shown in hexadecimal notation.

**Length** - The length in bytes of the data item. If the data item is a group whose size varies, the attributes show "group, size vary", and the length shown is the maximum size.

**Sect**-the section in which the data item was defined is indicated as follows:

    W-S- Working Storage section

    File-File section

    Link - Linkage section

**Align** - The alignment of the item is shown as follows:

    char- Byte alignment

    half- Halfword alignment

    full - Fullword alignment

**Attributes**- The type of the data item is shown as determined from the PICTURE, USAGE, SIGN, and other clauses. If the item is defined as EXTERNAL, "ext" precedes the type.

## Enable Fast (Native Mode) Binary

Keyword: FBIN            Values: YES   NO  
Command Line:            Omit   -f

*Note: Fast Binary is not currently available with COBOL ReSource.*

If FBIN = NO, the compiler generates code so that BINARY items are stored, and they are truncated so that the value fits within the number of decimal digits described by the PICTURE clause.

If FBIN = YES, the compiler generates code so a BINARY item can occupy the full range of values allowed by a fullword or halfword. FBIN is ignored if the item is unsigned.

A program executes more efficiently when FBIN = YES.

## Indexed Files Organization

Keyword: FILES        Values: DMS XDMS ANY

Command Line:        -DO   -D+   Omit

On COBOL ReSource, this option can be used to restrict certain features in program syntax. On the VS, this option provides a default for files specifying ORGANIZATION IS INDEXED without the DMS or XDMS phrase.

On the VS, if FILES=ANY, and the ORGANIZATION clause does not specify DMS or XDMS, the GENEDIT option determines whether indexed files are created as Indexed or Indexed-Plus files.

## FIPS Flag Level

Keyword: FIPS        Values: EXT   INT   LOW   NO

Command Line:        - j3   j2   jl   Omit

FIPS is an acronym for Federal Information Processing Standard. The FIPS Flagger indicates those source statements that contain syntax that is either nonstandard (Wang extensions) or above a specified level for federal Standard COBOL 85.

If FIPS = NO, no messages are generated. This is the default.

If FIPS = LOW, statements above low level and nonstandard (Wang extension) statements are flagged.

If FIPS = INT, statements above intermediate level and nonstandard (Wang extension) statements are flagged.

If FIPS = EXT, nonstandard (Wang extension) statements are flagged.

## Lowest Severity Error Printed

Keyword: FLAG      Value: n

Command Line:

An error with a severity code greater than or equal to n causes the compiler to print a diagnostic message. The error severities are

**0 FIPS Monitor**- This severity is used by the flagging mechanisms that are invoked by the FIPS and the OBS options but not controlled by the FLAG option.

**1 Information** - The program is correct, but the compiler has a suggestion to make the program more efficient.

**2 Caution** - The compiler has detected a possible error.

**4 Warning**- The compiler has detected an error but has probably generated the intended object code. It is recommended that these problems be fixed.

**6 Severe Warning**- The compiler has detected an error, and the generated object code is likely to function in a different manner from that intended. It is strongly recommended that these problems be fixed.

**8 Error** - The compiler has detected an error and cannot generate object code for this compilation.

**12 Severe Error** - The compiler has detected a situation that requires that the compilation be terminated immediately.

**16** - Compilation canceled by user from Options screen.

## Runtime Libraries to Link

Keywords:	LINKFILE	Values: file-name
	LINKLIB	library-name
	LINKVOL	volume-name
	MORE	YES NO
Command Line:		-L

*Note: Under the ReSource Command Line, the user cannot link against runtime libraries. By default, the Linker is run at the end of compilation to link against the ReSource Runtime. The -c option stops this link.*

The code generated for some COBOL 85 statements calls routines in the runtime library that must be linked in at some point. This can be done as part of the compilation. When multiple compilation units are to be linked to form a run unit, the runtime routines should be linked in. Refer to Appendix K for more information on link editing and the COBOL runtime.

If LINKFILE and LINKLIB are blank, the Linker is not run as part of the compilation process. Otherwise, the Linker is run, and these values specify the library in which the runtime routines reside.

The MORE option specifies whether additional routines reside in other libraries. If MORE=NO, the library specified in LINKVOL and LINKLIB is the only library searched. If you specify MORE = YES, the LINKLIBS screen appears. You may specify up to eight libraries for the Linker to search. The libraries are searched in the order in which they are specified.

The defaults are

LINKFILE (if supported by your compiler version) = @CBLRTF@

LINKLIB = @CBLRTM@

LINKVOL = your system volume

MORE = NO

## Generate Object Program

Keyword: LOAD      Values: NO    YES

Command Line:      -G      -o [pathname] or Omit

COBOL ReSource generates object code in the pathname for the -o option. When the -o option is omitted, the name of the source file to be compiled must be

name.cob

The compiler then produces an executable of name with no suffix. If the source file does not have the .cob suffix, it gets replaced by the resulting executable.

If LOAD = YES, the compiler creates an object program and stores it in an output file. If LOAD = NO, no object program is produced, and the compiler performs as a Lexical Syntax and Semantic verifier.

## Allow Lowercase Display and Read Input

Keyword: LOWER    Values: YES   NO

Command Line:      -w      Omit

If LOWER = YES, the default FAC generated for alphanumeric fields to be entered in response to a DISPLAY AND READ statement is to be hex 80 and accepts both uppercase and lowercase input. If LOWER = NO, the default FAC is hex 81 and causes lowercase input to shift to uppercase. This feature allows the replacement of the lowercase characters with a non-Latin character set. Such replacement is useful for international applications.

## Maximum Error Count

Keyword: MAXERRS      Value: n

Command Line:      -Mn

The parameter n specifies the number of level 8 errors at which compilation should be terminated. If, during compilation, the number of level 8 errors reaches the number specified in MAXERRS, compilation is terminated. The default is 100.

## MORE

Refer to the keywords LINKFILE, LINKLIB, LINKVOL.

## Object Module Format

Keyword: OBJFORM Values: 0 1

Command Line: Not supported

Object format 1 is the default. Two comments of importance are associated with this option:

- If OBJFORM = 0, only the first seven characters of the called program's name are significant and therefore must be unique. If OBJFORM = 1, the first 30 characters are significant and must be unique.
- With object format 0 object, anything in WORKING-STORAGE beyond one megabyte cannot be initialized. If more than one module is linked, this 1-megabyte limit applies to the run unit totality, not to each module individually.

## Flag Obsolete Elements

Keyword: OBS Values: NO YES

Command Line: -b omit

IF OBS = YES, any usage that has been designated as obsolete (that is, syntax that will not be supported in the next revision of the COBOL Standard) is flagged

## Optimize Object Code

Keyword: OBJFORM Values: NO 1 2 YES

Command Line: Omit -O1 -O2 -O

*Note: Optimization of the Object Code is not currently supported with COBOL ReSource.*

This option specifies whether the object code is to be optimized to improve execution speed. A program should not be optimized until it is ready for production use because some optimizations rearrange operations in the program and may cause confusion in debugging.

The degree of object code execution speed improvement from level to level depends on the characteristics of the source code and can vary considerably from program to program.

The optimizations performed for the various levels are listed as follows:

### *Level 1 Optimization*

- I. Pattern replacement. Various intermediate code patterns are replaced with more efficient patterns.
2. Constant folding. Expressions are replaced by their values, if those values can be computed at compile time.
3. Loop optimization. Loop invariant computations are moved out of loops to speed up loop execution.
4. Common subexpression elimination. Redundant computations are eliminated
5. Value propagation. References to variables are replaced by references to their assigned values.



6. Local unreadable code elimination. Unreachable code as determined by local code analysis is eliminated
7. Local assignment elimination. Redundant assignments as determined by local program analysis are eliminated.

#### *Level 2 Optimization*

1. All level 1 optimizations.
2. Global unreachable code elimination. More program control flow analyses are performed to identify unreachable code in the program. The unreachable code is eliminated.
3. Global assignment elimination. More program control flow analyses are performed to determine redundant assignments in the program. The redundant assignments are eliminated.

## **YES -- Level 3 Optimization**

1. **All** level 2 optimizations.
2. **PERFORM** statement inline expansion. A target procedure of a **PERFORM** statement is expanded inline if the procedure consists of a single paragraph, meets certain size constraints, and does not contain unsafe branches. The procedure reduces program branching and paging.

## **Memory Space Requirements for Global Optimization**

The global optimizer operates on a complete source program at a time. The memory requirement for the optimizer largely depends on the symbol table space used and the number of intermediate operators produced for the program. Level 2 and full optimization require more memory than level 1 optimization due to the increase in data flow analysis. The global optimizer performs two memory checks: the first before attempting level 1 optimization and the second before attempting level 2 and/or full optimization.

The global optimizer handles oversize programs as follows:

- If the specified **OPT** option is 1, 2, or **YES**, and the memory requirement for level 1 optimization exceeds the currently available memory, the optimizer issues the severe warning:

`The program is too large; no optimization was performed for this program.`

- If the specified **OPT** option is 2, or **YES**, and upon the completion of level 1 optimization, if the memory requirement for the rest of level 2 and full optimization exceeds the currently available memory, the optimizer issues the severe warning:

`The program is too big; only level 1 optimization was performed for this program.`

The compilation then resumes with the next phase of the compilation.

## **Print Program Map**

Keyword: **PMAP**      Values: **YES**   **RANGE**      **NO**

Command Line:      **-A**      **-A [lo] [:hi]**      **Omit**

If **PMAP** = **YES**, a **PMAP** is generated for the entire program. IF **PMAP** = **RANGE**, the **Ranges** screen enables you to specify which listing lines the **PMAP** should be generated for.

The program map consists of four basic columns:

Column 1- Line numbers

Column 2 - Address and machine instruction generated (hex values)

Column 3 - COBOL 85 paragraph names

Column 4 - Assembler instructions

## PACE COBOL Host Language Interface (HLI)

Keyword: PREP      Values: PACE Rev#   NO

Command Line:      Not supported

The PREP option enables the preprocessing of a COBOL 85 source file containing HLI syntax. The preprocessor converts HLI syntax to COBOL 85 syntax.

If PREP = NO, no preprocessing is done. If PREP = PACE Rev#, the Rev# is a 3-digit byte number that specifies preprocessing by WCHPxxx (for example, entering 200 results in preprocessing by WCHP200, the preprocessor for PACE 2.0). The PACE Rev# should be specified for VS PACE, Release 2.0 and greater.

Selecting PACE Rev# causes the Options screen prompt to allow users to indicate how much of the code generated by HLI syntax to print on a source listing. If SOURCE = NO is selected, any selection on the Options screen is ignored.

## Print COPY Files

Keyword: PRTCOPY   Values: YES   NO

Command Line:      Omit   -n

If PRTCOPY = YES, the contents of COPY files are printed on the listing. If PRTCOPY=NO, COPY statements are printed but COPY files are not.

## R-Margin Setting

Keyword: RMARGIN   Values: 72      80

Command Line:      Omit   -r

If MARGIN = 72, columns 73 through 80 in the source file are ignored by the compiler. If MARGIN = 80, columns 73 through 80 of each line are processed as part of the source.

## Separate Sign Character

Keyword: SEPSGN    Values: YES   NO

Command Line: Omit -h

The SEPSGN option specifies the sign characteristics (that is, whether the sign is to be associated with a digit position or appear as a separate byte) for usage display data items that do not have the SEPARATE phrase of the SIGN clause coded in the program. If SEPSGN = YES (the default), the sign is put in a separate, trailing byte. If SEPSGN = NO, the sign is included in a digit position.

## Print Source Listing

Keyword: SOURCE    Value: YES   NO

Command Line:      Omit -1      (lowercase L, *not* the numeral 1)

If SOURCE = YES, the compiler produces a source listing of the compiled program with accompanying diagnostics. If SOURCE = NO, source lines are not printed in the compile listing.

## Code Generation Stop Level

Keyword: STOP      Value: n

Command Line:      -un

The parameter n specifies the lowest level of error severity that will cause the compiler to abort the compilation. Any error with a severity code greater than or equal to n will terminate the compilation, no object program being produced. If n is greater than eight, eight is used. The default is eight.

## Check Subscripts

Keyword: SUBCHK    Values: YES   NO

Command Line:      -s      Omit

*Note: Subscripts Check is not currently supported with COBOL Resource.*

If SUBCHK = YES, the compiler generates special code that checks the ranges of subscripts during program execution. If a subscript exceeds its defined limit, a program check (execution interruption) is performed. This results in a larger, slower program. The default is NO.

## Symbolic Debug Information

Keyword: SYMB      Values: YES   NO

Command Line:      -g      Omit

If SYMB = YES, the compiler records symbolic debug information in the object program. If SYMB = NO, this information is not retained.

## Truncate COMP to PIC

Keyword: TRUNC    Values: YES   NO

Command Line:      -t      Omit

If TRUNC = YES, truncation will occur for PACKED DECIMAL (COMP) data in cases when a larger numeric data item is moved into a smaller numeric data item.

If the receiving field does not have enough bytes to contain all the digits from the sending field, truncation of one or more high-order digits is automatic. This situation is not altered by TRUNC.

In the special case involving packed decimal fields, however, truncation can be specified even if the receiving field has enough bytes to accommodate the sending field.

This case occurs when the receiving field has an even number of digits (for example, PICTURE S9(6) COMP). Each digit occupies a half-byte in packed decimal format. Since the low-order half-byte is taken by the sign digit, the number of half-bytes available for digits is always odd. If the number of digits is even, the high-order half-byte in the field is unused and is set to 0. For example, the packed decimal field generated by a data item with a PICTURE of S9(6) COMP containing the value 123456 looks like 0123 45 6F, where each pair of digits occupies one byte, and the F is the sign digit.

Note that although the PICTURE specifies six digits, the resulting value occupies four bytes in memory. If TRUNC = YES, and a numeric data item of seven digits (or more) is moved

into this field, the leftmost digit is truncated, even though the receiving field contains an available half-byte to store it. If TRUNC = NO (the default value), the leftmost digit is placed in the high-order half-byte of the receiving field. If the receiving field contains more than seven digits, any digits beyond the seventh are truncated automatically, irrespective of the value of TRUNC.

## Print Cross-Reference Information

Keyword: XREF      Values: YES   NO

Command Line:      -x      Omit

If XREF = YES, an alphabetical cross-reference listing of all data-names referenced in the program is produced. This listing contains each data-name, the source program line on which it is initially defined, and all other lines in the program in which the data-name is referenced.

## Performance Considerations

By using the coding practices in the following list and Compiler option settings described in the next section, the user can take full advantage of the compiler's optimizing capabilities:

- The user must always use binary items for subscripts and for loop counters.
- The user must avoid using DISPLAY items as subscripts, loop counters, or using them in any context that requires numeric calculation. Since DISPLAY is the default usage, this recommends additional coding USAGE clauses.
- The user must use the SYNC clause on binary items.

## Option Settings

The following Compiler option settings will increase the program's execution speed:

- COMPAT74 = IO offers the greatest compatibility with the VS DMS. It omits the extra checking required to return the new COBOL 85 status codes. Access to numeric data is faster than with COMPAT74 = YES.
- SEPSGN = NO allows faster access to signed numeric data with usage DISPLAY than when the sign is separate.
- FBIN = YES causes binary items to be treated as native binary. No truncation to PICTURE is performed.
- OPT = YES runs the optimizer.

## Compiler Directives

Compiler directives appear in a program's source code. They direct the compiler to perform certain actions.

## Debug Control

To support debugging of VS PACE COBOL HLI preprocessed code, the following directives may be emitted by the HLI preprocessor and are recognized by the compiler.

### **DEBUG-UNIT-BEGIN**

This directive defines the beginning of a debug unit. A debug unit consists of one or more lines that, during debugging, execute as an indivisible unit, as if they were a single line of code. Setting a trap within a debug unit will result in a trap being taken at the first line of the debug unit, and the next single step will go to the first line following the end of the debug unit. The "\$" must appear in column 7 and must be immediately followed, with no intervening spaces, by "DEBUG-UNIT-BEGIN".

### **DEBUG-UNIT-END**

This directive defines the end of a debug unit. The "\$" must appear in column 7 and must be immediately followed, with no intervening spaces, by "DEBUG-UNIT-END".

# COPY Statement

The COPY statement incorporates library text into a source program, of which it is then considered a part.

## Format

$$\begin{array}{l} \text{COPY} \left\{ \begin{array}{l} \text{file-name-1} \\ \text{literal-1} \end{array} \right\} \left[ \left\{ \begin{array}{l} \text{OF} \\ \text{IN} \end{array} \right\} \left\{ \begin{array}{l} \text{library-name-1} \\ \text{literal-2} \end{array} \right\} \left[ \left\{ \begin{array}{l} \text{OF} \\ \text{IN} \\ \text{ON} \end{array} \right\} \left\{ \begin{array}{l} \text{volume-name-1} \\ \text{literal-3} \end{array} \right\} \right] \right] \\ [ \text{SUPPRESS} ] \\ \left[ \text{REPLACING} \left\{ \begin{array}{l} \text{==pseudo-text-1==} \\ \text{identifier-1} \\ \text{literal-4} \\ \text{text-word-1} \end{array} \right\} \text{BY} \left\{ \begin{array}{l} \text{==pseudo-text-2==} \\ \text{identifier-2} \\ \text{literal-5} \\ \text{text-word-2} \end{array} \right\} \dots \right] . \end{array}$$

## Syntax Rules

1. The COPY statement must be preceded by a space and terminated by a period.
2. The COPY statement can appear anywhere within a source program in which a character-string or separator (other than a closing quotation mark) can occur, except within another COPY statement.
3. When more than one library is available during compilation, file-name-1 must be qualified by either library-name-1 or volume-name-1 unless the Compiler option COPYLIBS is YES.
4. File-name-1, library-name-1, or volume-name-1 can be any name recognized by the file system as a valid file, library, or volume name, respectively.
5. The value of literal-1, literal-2, or literal-3 can be any name recognized by the file system as a valid file, library, or volume name, respectively.
6. Text words within a library or within pseudo-text can be from 1 to 322 characters in length.
7. Pseudo-text-1 cannot be just a comma or just a semicolon.
8. Text-word-1 and text-word-2 can be any single COBOL word except COPY.
9. If the word COPY is in a comment-entry or in a place where a commententry can appear, it is considered part of the comment-entry.

## General Rules

1. **All** COPY statements are processed before the source program is compiled. Therefore, the syntactic correctness of the source program cannot be determined until these statements have been processed.
2. Processing a COPY statement consists of copying the specified library text into the source program and logically replacing the entire COPY statement. The text replaced begins with the word COPY and ends with the terminating period
3. If the REPLACING phrase is not specified, the library text is copied unchanged. If the REPLACING phrase is specified, each properly matched occurrence of

pseudo-text-1, identifier-1, word-1, or literal-4 in the library text is replaced by the corresponding pseudo-text-2, identifier-2, word-2, or literal-5 when the library text is copied.

4. If the REPLACING phrase is specified, the rules for matching items for replacement are
  - The first word in the library text that is considered for matching is the leftmost word that is not a separator comma or a separator semicolon. (Any text word or space preceding this word is simply copied into the source program.) The operand that precedes the word BY in the first BY phrase is compared to the same number of contiguous library text words, beginning with the first text word, as defined above.
  - The operand matches the library text if the text words that it contains are equal, character for character, to the library text words with which it is being compared. However, for purposes of matching, a separator comma or semicolon in pseudo-text-1 or in the library text is considered to be a single space, as are two or more consecutive spaces. Thus, the sequence (space) (space) (comma) (semicolon) (space) is considered a single space.
  - If the two items do not match, the above comparison is repeated with the operand preceding BY in the next BY phrase. This process continues until either a match occurs or the operands in all the BY phrases have been compared with the word without a match having occurred.
  - If the operands in all the BY phrases have been compared and no match has been found, then the leftmost library text word is copied into the source program. The next successive eligible library text word (that is, a word that is not a separator comma or semicolon) is now considered the leftmost, and the comparison process begins again with the operand preceding BY in the first BY phrase.
  - If a match is found, the corresponding operand (pseudo-text-2, identifier-2, word-2 or literal-2) is copied into the source program. Then, the library text word immediately following the rightmost text word that was matched becomes the new leftmost text word, and the comparison process begins again with the first operand in the first BY phrase.
  - These comparisons continue until the rightmost text word in the library text has either been matched or has been compared as the current leftmost library text word with all the operands preceding BY in all the BY phrases.
5. For purposes of matching, identifier-1, and literal-4 are treated as pseudotext. That is, the name is what is important, not the contents of the corresponding object.
6. Any comment or blank lines in the library text or in pseudo-text-1 are ignored for purposes of matching. Whenever pseudo-text-2 is copied into the source program as a result of text replacement, any comment or blank lines that it contains are also copied. Otherwise, any comment or blank lines in the library text are copied into the



source program unchanged, except that a comment or blank line within a sequence of text words that matches pseudotext-1 is not copied.

7. A text word that is copied from a library and not replaced is generally copied so as to start in the same area of the program line as it begins in the line in the library. The one exception is when a text word to be copied (or replaced) begins in area A and follows a text word that also begins in area A and is being replaced by a longer word. In this case, if there is no room left in area A, the second word will begin in area B.
8. Each text word in pseudo-text-2 that is copied into the source program is copied beginning in the same area as that in which it appears in pseudotext-2.
9. Each identifier-2, literal-2 and text-word-2 copied into the source program is copied beginning in the same area as that in which the leftmost library word that it matched appeared.
10. Library text must conform to the rules for COBOL 85 reference format.
11. Debugging lines are permitted in both library text and in pseudo-text. In a debugging line, text words are compared as if the 'D' did not appear in the indicator area of the line.
12. If the COPY statement begins on a debugging line, or if the text word is on a debugging line in the library, any text words that are copied will be put debugging lines. When a text word specified in a BY phrase is copied, it will be on a debugging line if the first library text word being replaced was on a debugging line.
13. Except for the cases in rule 12, only those text words that are specified on debugging lines contained in pseudo-text-2 will be on debugging lines in the program.
14. If a literal that is specified as literal-2, in pseudo-text-2, or as library text, is too long to fit on a single line but is not being copied on a debugging line, continuation lines will be created to contain the remainder of the literal. (If the specified replacement requires that the literal be continued on a debugging line, the program is in error.)
15. For purposes of compilation, text words after replacement are copied into the source program according to the rules for the reference format. When pseudo-text-2 is copied, additional spaces can be introduced only between text words that are already separated by a space (this includes the implicit space between source lines).
16. If additional lines are introduced into the source program by action of a COPY statement, the indicator area of each new line will contain the same character as does the line in which the text being replaced begins, unless this line contains a hyphen. In this case, each new line will contain a space in the indicator area. Furthermore, if a literal is continued on an introduced line that is not a debugging line, the new line will contain a space in the indicator area.
17. If a file-name and library-name are specified but a volume-name is not, the default volume name INVOL is used.
18. If only a file-name is specified and the Compiler option COPYLIBS is YES, the search for the file takes place in the libraries and volumes specified in the COPYLIBS option. If only a file-name is specified and COPYLIBS is NO, the default library, INLIB, and default volume, INVOL, are used. If the file is not found,

the system will issue a prompt to the workstation, requesting the file, library, and volume names.

19. If SUPPRESS is specified, the library text is not printed in the source program listing.
20. If the REPLACING phrase is specified, the library text should not contain a COPY statement, and the text resulting from processing the REPLACING phrase must not contain a COPY statement.
21. If the REPLACING phrase is not specified, the library text may contain a COPY statement that does not have a REPLACING phrase. Such a COPY statement is said to be a nested COPY statement. COPY statements can be nested up to five deep.

# REPLACE Statement

The REPLACE statement replaces text in the source program with other text.

## *Format 1*

```
REPLACE {==pseudo-text-1== BY ==pseudo-text-2==} ... .
```

## *Format 2*

```
REPLACE OFF .
```

## *Syntax Rules*

1. A REPLACE statement can appear anywhere in the source program in which a character-string can appear.
2. The REPLACE statement must be in a sentence by itself.
3. Pseudo-text-1 must contain at least one text word.
4. Pseudo-text-2 can be null, or it can contain one or more text words.
5. Character-strings in pseudo-text-1 and pseudo-text-2 can be continued.
6. A text word in pseudo-text-1 or pseudo-text-2 can be from 1 to 322 characters long.
7. Pseudo-text-1 cannot consist of only a separator comma or a separator semicolon.
8. If the word REPLACE is found in a comment-entry or where a commententry can appear, it is treated as part of the comment-entry.

## *General Rules*

1. A format 1 REPLACE statement replaces each matched occurrence of pseudo-text-1 in the source program by pseudo-text-2.
2. A REPLACE statement is in effect from the point at *which* it occurs in the source program until either another REPLACE statement or the end of the separately compiled program occurs. (This is referred to as the scope of the REPLACE statement.)
3. A format 2 REPLACE statement turns off the operation of a preceding format 1 REPLACE statement.
4. REPLACE statements are processed after all COPY statements have been processed.
5. Pseudo-text-2 cannot contain a REPLACE or COPY statement.
6. Items are matched for replacement as follows:
  - Starting with the leftmost source program text word following the REPLACE statement, and with the first pseudo-text-1 entry, pseudotext-1 is compared to an equal number of contiguous text words in the source program.
  - The operand matches the library text if the text words that it contains are equal, character for character, to the library text words with which it is being compared. However, for purposes of matching, a separator comma or semicolon in pseudo-text- I or in the library text is considered to be a single

space, as are two or more consecutive spaces. Thus, the sequence (space) (space) (comma) (semicolon) (space) is considered a single space.

- If a match is not made, this comparison is repeated with each successive pseudo-text-1 in the statement until either a match is made or all the pseudo-text-1s have been compared. (Note that since these pseudo-texts need not contain the same number of words, each matching operation can be with a different number of words in the source text.)
  - If all the pseudo-texts have been compared and no match has occurred, the next successive source text word is now considered the leftmost word, and the comparison process described previously begins again with the first pseudo-text-1.
  - When a match occurs, the pseudo-text-2 associated with the matched pseudo-text-1 is copied into the source program, replacing it. At this point, the next source text word following the rightmost of the text words that were matched is considered the leftmost word, and the comparison begins again with the first pseudo-text-1.
  - These comparisons continue until the rightmost text word in the source program text within the scope of the REPLACE statement has either been part of a match, or has participated in a comparison with all the pseudo-text-1s.
7. Text words put into a source program by a REPLACE statement are inserted according to the rules for the reference format. When a pseudo-text-2 is copied, additional spaces can be introduced only between text words that are already separated by a space (this includes the implicit space between source lines).
  8. If additional lines are introduced into a source program by a REPLACE statement, the indicator area of each new line will contain the same character as does the line in which the text being replaced begins, unless this line contains a hyphen. In that case, each new line will contain a space in the indicator area.
  9. Any comment or blank lines found in the source text or in pseudo-text-1 are ignored for purposes of matching. The sequence of text words in the source program text and in pseudo-text-1 is determined by the rules for the reference format. Whenever pseudo-text-2 is copied into the source program as a result of text replacement, any comment or blank lines that appear in it are copied also. A comment or blank line within a sequence of source text words that match pseudo-text-1 is not replaced.

10. If a literal in pseudo-text-2 is too long to fit on a single line, and the literal is not being copied to a debugging line, continuation lines are created, as necessary, to accommodate the literal. If the replacement is such that the literal would have to be continued on a debugging line, the program is in error.
11. Debugging lines are permitted in pseudo-text. Within a debugging line, text words are compared as if the D did not appear in the indicator area.
12. All REPLACE (and COPY) statements are processed before the source program is compiled. Therefore, the syntactic correctness of the source program cannot be determined until all COPY and REPLACE statements have been processed

# Appendix C Field Attribute Characters

## Introduction

An important part of the workstation screen is its ability to be divided into fields. A field is defined as all of the characters from one field attribute character to the next or to the end of the row. Field attribute characters control the operation of the workstation. All the characters of a field have the same attributes, which are defined by the field attribute character preceding the field. They occupy a location on the display and always appear as a blank no matter what their value. Any location of the screen can be a field attribute character.

The field attribute character controls the mode of display for all following characters until another field attribute character or until the end of the row is encountered. Each row is considered to have a field attribute character just before the first character in the row and just after the last character in the row. These field attribute characters, which have a default value of hex 8C, do not take up space on the screen, allowing a user to use the full 80-character width of each line.

The possible attributes are defined in the following table:

<b><u>Bit</u></b>	<b><u>Field Description</u></b>
0	Must be 1. This is the field attribute character indicator.
1	Modified data tag = 0 Field has not been modified. = 1 Field has been modified.'
2	Underscore option = 0 Field is not underscored. = 1 Field is underscored.
3,4	Display control = 00 Intensified = 01 Low intensity = 10 Blinking = 11 Nondisplay
5	Protect bit = 0 Modifiable field = 1 Protected field
6,7	Valid keyable data specification = 00 Alphanumeric = 01 Uppercase = 10 Numeric only = 11 Reserved

## Display Characteristics for Workstation Screen Fields

The possible display characteristics for fields of the workstation screen are

**Intensified display** - The characters in this field are displayed in higher intensity than those in a low-intensity display field.

**Low intensity display** - The characters of this field are displayed on the screen at normal intensity.

**Blinking** - The characters in this field are displayed alternately in Intensified Display and Normal Display mode. The display changes modes at a fixed rate of about three times a second.

**Nondisplay** - The characters in the field are not displayed on the screen. The field looks as if it is all blanks.

**Modifiable** - Any or all of the positions of this field can be changed by the operator.

**Protected** - No position of this field can be modified by the operator.

**Alphanumeric** - Any character on the keyboard can be keyed in.

**Uppercase** - Letters are displayed and stored only as uppercase. This is without regard to whether SHIFT or LOCK is depressed. All other keys respond to the SHIFT and LOCK normally.

**Numeric** - Only the characters 0 to 9, decimal point (.), and minus sign (-) may be entered into this field, or the keystroke is ignored and the alarm sounds.

**Reserved** - This is not a valid combination at this time. It is intended for the addition of later options. Its use can result in unpredictable results.

The following table provides a list of the field attribute characters:

<b>Intensity</b>	<b>Input Control</b>	<b>Characters</b>	<b>Underline</b>	<b>Hex Value</b>
BRIGHT	MODIFY	ALL	NOLINE	80
BRIGHT	MODIFY	UPPERCASE	NOLINE	81
BRIGHT	MODIFY	NUMERIC	NOLINE	82
BRIGHT	PROTECT	ALL	NOLINE	84
BRIGHT	PROTECT	UPPERCASE	NOLINE	85
BRIGHT	PROTECT	NUMERIC	NOLINE	86
DIM	MODIFY	ALL	NOLINE	88
DIM	MODIFY	UPPERCASE	NOLINE	89
DIM	MODIFY	NUMERIC	NOLINE	8A
DIM	PROTECT	ALL	NOLINE	8C
DIM	PROTECT	UPPERCASE	NOLINE	8D
DIM	PROTECT	NUMERIC	NOLINE	8E

(continued)



Intensity	Input Control	Characters	Underline	Hex Value
BLINK	MODIFY	ALL	NOLINE	90
BLINK	MODIFY	UPPERCASE	NOLINE	91
BLINK	MODIFY	NUMERIC	NOLINE	92
BLINK	PROTECT	ALL	NOLINE	94
BLINK	PROTECT	UPPERCASE	NOLINE	95
BLINK	PROTECT	NUMERIC	NOLINE	96
BLANK	MODIFY	ALL	NOLINE	98
BLANK	MODIFY	UPPERCASE	NOLINE	99
BLANK	MODIFY	NUMERIC	NOLINE	9A
BLANK	PROTECT	ALL	NOLINE	9C
BLANK	PROTECT	UPPERCASE	NOLINE	9D
BLANK	PROTECT	NUMERIC	NOLINE	9E
BRIGHT	MODIFY	ALL	LINE	A0
BRIGHT	MODIFY	UPPERCASE	LINE	A1
BRIGHT	MODIFY	NUMERIC	LINE	A2
BRIGHT	PROTECT	ALL	LINE	A4
BRIGHT	PROTECT	UPPERCASE	LINE	A5
BRIGHT	PROTECT	NUMERIC	LINE	A6

(continued)

Intensity	Input Control	Characters	Underline	Hex Value
DIM	MODIFY	ALL	LINE	A8
DIM	MODIFY	UPPERCASE	LINE	A9
DIM	MODIFY	NUMERIC	LINE	AA
DIM	PROTECT	ALL	LINE	AC
DIM	PROTECT	UPPERCASE	LINE	AD
DIM	PROTECT	NUMERIC	LINE	AE
BLINK	MODIFY	ALL	LINE	B0
BLINK	MODIFY	UPPERCASE	LINE	B1
BLINK	MODIFY	NUMERIC	LINE	B2
BLINK	PROTECT	ALL	LINE	B4
BLINK	PROTECT	UPPERCASE	LINE	B5
BLINK	PROTECT	NUMERIC	LINE	B6
BLANK	MODIFY	ALL	LINE	B8
BLANK	MODIFY	UPPERCASE	LINE	B9
BLANK	MODIFY	NUMERIC	LINE	BA
BLANK	PROTECT	ALL	LINE	BC
BLANK	PROTECT	UPPERCASE	LINE	BD
BLANK	PROTECT	NUMERIC	LINE	BE

## Appendix D

### Workstation Screen Order Area

#### Introduction

The order area bytes are used to specify screen control actions and to indicate row/column addresses for data transfer to and from the display.

The content of the order area and the interpretation of the fields in the area is different for the READ and REWRITE statements. The following list illustrates the differences:

<b>Byte</b>	<b>On READ</b>	<b>On REWRITE</b>
<b>0</b>	<b>Row number</b>	<b>Row number</b>
<b>1</b>	<b>Reserved</b>	<b>Write Control Character (WCC)</b>
<b>2</b>	<b>Cursor Column Address</b>	<b>Cursor Column Address</b>
<b>3</b>	<b>Cursor Row Address</b>	<b>Cursor Row Address</b>

The row number is the starting screen line number for data transfer. The byte is set to the value specified for the data-name of the RELATIVE KEY phrase.

## Interpretation of the Order Area on a Rewrite

Neither the order area nor the mapping area is changed on a REWRITE operation. The first byte of the order area on a REWRITE is interpreted as the row number at which the REWRITE is to start. If this row number is not in the range 1 through 24, the command is terminated

The second byte of the order area is interpreted as the WCC. The WCCs are interpreted as follows:

Bit	Explanation
0	UNLOCK keyboard (LOCK is zero)
1	Sound alarm
2	Position cursor
3	Roll down
4	Roll up
5	ERASE rest of modifiable fields
6	ERASE and protect rest of screen
7	Reserved (must be zero)

### Unlock the Keyboard (Hex "80")

After the record is written to the screen, and after the alarm is sounded, if this command is specified, the AID character is set to blank, and the keyboard is then unlocked.

If the bit is zero, the keyboard is locked before any data is transmitted to the workstation. If the keyboard is already locked, setting this bit to zero does not change the status of the keyboard. The normal method to get the keyboard locked is to wait for the operator to press one of the computer communication keys. If the bit is zero and the keyboard is locked, the AID character does not change. However, if the command locks the keyboard, the AID character is set to an apostrophe (Hex "21 ").

### Sound Alarm (Hex "40")

This command causes the alarm to sound before the data is transmitted to the screen.

## **Position the Cursor (Hex "20")**

The cursor row address byte must have a value between 0 and 24 inclusive, and the cursor column address byte must have a value between 0 and 80 inclusive. After the Write completes, the cursor is positioned at that row and column. If the cursor row address byte is 0, it is treated as if it were 1. If the cursor column address byte is 0, this acts as if the cursor were positioned one location before the first location in the specified row and the TAB key were pressed. If there are no modifiable positions on the screen after the WRITE command, the cursor is positioned at the first location in the specified row.

If the cursor position bit is set in the WCC and the cursor row address byte has a value other than 0 to 24 or the cursor column address byte has a value other than 0 to 80, the command is terminated with an error.

*Note: This address can have values from 1 to 80.*

## **Roll Down (Hex "10")**

This command causes the bottom line of the screen to be lost and each line above it to be copied into the next lower line. This proceeds until the row specified in the order area is copied. The specified row is then set to blanks and the WRITE operation proceeds. An attempt to write more than one line in a single command with "roll down" results in an error.

## **Roll Up (Hex "08")**

This command causes the row specified in the order area to be lost and each line below it to be copied into the next higher line (for example, line 1 is replaced by the contents of line 2). This proceeds until the last row of the screen is copied. The last row is then set to blanks, and the WRITE operation proceeds on the last line of the screen. An attempt to write more than one line in a single command with "**roll** up" specified results in an error.

## **Erase Modifiable Fields to Pseudo Blanks (Hex "04")**

Modifiable locations starting at the row address specified in the order area to the end of the screen are set to pseudo-blank characters (Hex 0B).

## **Erase and Protect Rest of Screen (Hex "02")**

All locations of the screen at and after the row address specified in the order area to the end of the screen are set to the hex "8C" before the data is transferred to the screen; thus there are no modifiable locations after the data that is written.

## Interpretation of the Order Area on a Read

The first byte of the order area is inspected before the data transfer and is used to specify the starting row number for the READ. If this row number is not in the range 1 to 24 (binary), the command is terminated. This byte is not changed by the READ statement.

The third and fourth bytes of the order area are set by the READ to the address of the cursor at the time of the READ. The first byte of the two contains the row number 1 to 24 (binary), and the second contains the column number 1 to 80 (binary) of the current cursor location. These two bytes are not inspected before the READ.

The second byte of the order area is not inspected or modified but is supplied as binary zeros for compatibility with future options.

## Workstation Screen Mapping Area Control

The Workstation Screen mapping area is defined in the program in the area immediately after the order area. The mapping area contains the data transmitted either to or from the screen. The first location of the mapping area corresponds to the first character of the row specified in the first byte of the order area. Byte 81 of the mapping area corresponds to the first byte of the next row. If the starting row number and the length of the mapping area are such that locations in the mapping area extend past the end of the screen, the command is terminated. Note that, although the mapping area's first position always corresponds to the start of a row, the only restriction on the end of the mapping area is that it not extend past the end of the screen. It is recommended that the mapping area be the full screen (1,920 bytes) as smaller quantities may result in a noticeable flicker.

No mapping area need be supplied for a 4-byte READ or REWRITE of the order area. Refer to Appendix C for more information on the formatting of the workstation screen and the mapping area.

## Display Characters

The CRT screen is capable of displaying 24 rows of 80 characters each. Every position of the screen is capable of displaying any of the possible display characters. A special symbol, called a cursor, is displayed beneath a character position to indicate where the next character entered from the keyboard is stored. The cursor is displayed on the screen when data can be keyed by the operator. If it is not displayed, the keyboard is locked. This has no effect on the display or the computer interface with the workstation, but it does directly have an effect on the data entry from the keyboard. Each position of the screen is referenced by its row and column numbers. The first position of the screen (upper left corner) is called row 1, column 1. The columns are numbered from left to right and the rows from top to bottom. Position two is the second character from the left on the first line.

The characters in the following list constitute all of the display characters:

### **Display Characters**

(space)

!

"

#

\$

%

&

'

(

)

\*

+

-

.

/

0 1 2 3 4 5 6 7 8 9

:

;

<

=

>

]

@

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

[ (open bracket)

\ (back slash)

] (close bracket)

^ (circumflex)

\_ (under bar)

a b c d e f g h i j k l m n o p q r s t u v w x y z

◊ (lozenge)

■ (solid character)

# Appendix E File Status Key Values

## Introduction

When the FILE STATUS clause is specified in a file control entry, a value is placed into the specified 2-character data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, or START statement and before any applicable USE procedure is executed, to indicate to the COBOL program the status of that input/output operation.

There are two sets of file status codes, depending on the setting of the Compiler option COMPAT74: COMPAT74 = YES and COMPAT74 = NO.

File status codes are divided into subgroups by general significance. The leftmost character position of the FILE STATUS data item is known as status key 1; it indicates the file status code subgroup as follows:

**'0' Successful Completion** - The INPUT/OUTPUT statement was successfully executed.

**'1' AT END** - The sequential READ statement was unsuccessful due to an attempt to read a record when no next logical record exists in the file or when the first READ statement is executed for an optional file that was unavailable when it was opened.

**'2' INVALID KEY** - The INPUT/OUTPUT statement was unsuccessful because of an invalid key condition.

**'3' Permanent Error Condition** - The INPUT/OUTPUT statement was unsuccessful because of a boundary violation or as the result of an input/output error, such as a data check, parity error, or transmission error.

**'4' Logic Error** - The INPUT/OUTPUT statement was unsuccessful because of an improper sequence of input-output operations that were performed on the file or because a limit defined by the user was violated

**'6' 1-O Cancel Condition** - The cancel messages have been suppressed as a result of a user request.

**'7' TIMEOUT** - An attempt to execute an INPUT/OUTPUT statement in SHARED mode failed due to a timeout.

**'8' SHARED MODE Error** - An error occurred while the file was open in SHARED mode.

**'9' Special Conditions**- The INPUT/OUTPUT statement was unsuccessful as a result of a condition, such as an invalid function or function sequence, that is unique to the file or device.



## AID Characters

The Attention Identifier (AID) character is the rightmost character of the FILE STATUS data item when the leftmost character is set to 0 by the processing of a READ statement on a workstation (DISPLAY) device file.

The AID character is modified when the operator presses the Enter key, HELP key, and the Program Function keys. The following table illustrates the AID character values corresponding to each of the actions.

### AID Configurations

AID	Hex Character (ASCII)	Graphic Character	AID	Hex Character (ASCII)	Graphic Character
Keyboard unlocked	20	blank	Locked by write	21	
Enter	40	@			
PF1	41	A	PF17	61	a
PF2	42	B	PF18	62	b
PF3	43	C	PF19	63	c
PF4	44	D	PF20	64	d
PF5	45	E	PF21	65	e
PF6	46	F	PF22	66	f
PF7	47	G	PF23	67	g
PF8	48	H	PF24	68	h
PF9	49	I	PF25	69	i
PF10	4A	J	PF26	6A	j
PF11	4B	K	PF27	6B	k
PF12	4C	L	PF28	6C	l
PF13	4D	M	PF29	6D	m
PF14	4E	N	PF30	6E	n
PF key 15	4F	O	PF key 31	6F	o
PF key 16	50	P	PF key 32	70	p

# File Status Codes - Compiler Option COMPAT74 YES

An explanation of the file status codes is given in the next sections for COMPAT74 = YES.

## Sequential Files

The Sequential file status codes are described in the following sections.

### 0 Successful completion

00 No further information available.

### 1 AT END

10 No further information available.

11 End-of-volume has been reached for a tape file and the user program indicates no automatic volume switch.

### 2 INVALID KEY

23 The supplied relative record number is equal to or greater than the highest record number in the file.

### 3 Permanent Error Condition

30 A hardware error occurred. This value is not returned for program-related errors.

34 An attempt was made to write beyond the externally defined boundaries of a file.

35 Returned by an OPEN statement. In OUTPUT mode, indicates a duplicate filename; in non-OUTPUT mode, a library not found or a volume not mounted.

38 An attempt was made to open a file that has been closed with the LOCK clause specified.

39 Returned by an OPEN statement. A conflict was detected between the fixed file attributes specified for the file in the program. For example, an unacceptable tape label or density was specified for a tape file, there were insufficient user access rights, or the file format was different from that specified.

#### **4 Logic Error**

- 41 File possession conflict; an OPEN statement was attempted for a file in the OPEN mode.
- 44 Insufficient space in library for file of specified size to be opened in OUTPUT mode.

#### **6 I-O Cancel Condition**

- 60 No further information available.

#### **7 Timeout**

- 70 No further information available.

#### **8 SHARED Mode Error**

- 80 No further information available.
- 82 Unsuccessful attempt by the system to update a file label.
- 83 SHARED mode malfunction that can be corrected only by doing an Initial Program Load (IPL). Refer to the *VS System Operator's Guide*.
- 84 Attempt to REWRITE a variable-length record whose record length is greater than the maximum record size specified for the file.
- 85 Attempt to update a file for which the user has Read-only access.
- 86 Invalid sequence of function requests. For example, an attempt to do a START HOLD on a file while another file is already held.

## **9 Special Conditions**

- 95 Invalid function request or invalid sequence of function requests for a given combination of device type, OPEN mode, and file organization. For example, an attempt to WRITE a record while the file is opened in Input mode.
- 97 An attempt to execute an I-O operation for a record of invalid length.

## **Indexed Files**

The Indexed file status codes are described in the following sections.

### **0 Successful Completion**

- 00 No further information available.
- 02 Duplicate key in an alternate indexed file.

### **1 AT END**

- 10 No further information available.

### **2 INVALID KEY**

- 21 Sequence error for a sequentially accessed Indexed file. The ascending sequence requirements of successive record key values have been violated, or the record key value has been changed by the COBOL program between the successful execution of a READ statement and the execution of the next REWRITE statement for the file.
- 22 Duplicate record key value. An attempt was made to WRITE or REWRITE a record that would create a duplicate record key.
- 23 No record found. An attempt was made to access a record that does not exist.
- 24 Boundary violation. An attempt was made to WRITE beyond the externally defined boundaries of the file.

### **3 Permanent Error Condition**

- 30 A hardware error occurred. This value is not returned for program-related errors.
- 34 An attempt was made to write beyond the externally defined boundaries of a file.
- 35 Returned by OPEN statement. In OUTPUT mode, indicates a duplicate file name; in non-OUTPUT mode, a library not found or a volume not mounted

- 38 An attempt was made to open a file that has been closed with the LOCK clause specified.
- 39 Returned by an OPEN statement. A conflict was detected between the fixed file attributes specified for the file in the program. For example, an unacceptable tape label or density was specified for a tape file, there were insufficient user access rights, or the file format was different from that specified.

#### **4 Logic Error**

- 41 File possession conflict. An OPEN statement was attempted for a file in the OPEN mode.
- 44 Insufficient space in library or library VTOC for file of specified size to be opened in OUTPUT mode.

#### **6 I-O Cancel Condition**

- 60 No further information available.

#### **7 Timeout**

- 70 No further information available.

#### **8 Shared Mode Error**

- 80 No further information available.
- 82 Unsuccessful attempt by the system to update a file label.
- 83 SHARED mode malfunction that can be corrected only by doing an Initial Program Load (IPL). Refer to the *VS System Operator's Guide*.
- 84 Attempt to REWRITE a variable-length record whose record length is greater than the maximum record size specified for the file.
- 85 Attempt to update a file for which the user has Read-only access.
- 86 Invalid sequence of function requests. For example, an attempt to do a START HOLD on a file while another file is already held.

#### **9 Special Conditions**

- 95 Invalid function request or invalid sequence of function requests for a given combination of device type, OPEN mode, and file organization. For example, an attempt to WRITE a record while the file is opened in INPUT mode.

- 97     An attempt to execute an I-O operation for a record of invalid length.
- 98     Use of a nonexistent alternate record key when attempting to REWRITE or WRITE a record of an alternate-indexed file.

## Relative Files

The Relative file status codes are described in the following sections.

### **0 Successful completion**

- 00     No further information available.

### **1 AT END**

- 10     No further information available.

### **2 INVALID KEY**

- 22     Duplicate record key value. An attempt was made to WRITE or REWRITE a record that would create a duplicate record key.
- 23     No record found. An attempt was made to access a record that does not exist.
- 24     Boundary violation. An attempt was made to WRITE beyond the externally defined boundaries of the file, or a START has been issued for a record key greater than the highest record key value in the file.

### **3 Permanent Error Condition**

- 35     Returned by OPEN statement. In OUTPUT mode, indicates a duplicate filename; in non-OUTPUT mode, a library not found or a volume not mounted.
- 38     An attempt was made to open a file that has been closed with the LOCK clause specified.
- 39     Returned by OPEN statement. A conflict was detected between the fixed file attributes specified for the file in the program. For example, an unacceptable tape label or density was specified for a tape file, there were insufficient user access rights, or the file format was different from that specified

### **4 Logic Error**

- 41     File possession conflict. An OPEN statement was attempted for a file in the OPEN mode.
- 44     Insufficient space in library or library VTOC for file of specified size to be opened in OUTPUT mode.

## 9 Special Conditions

- 97      An attempt to execute an I-O operation for a record of invalid length.

## File Status Codes - Compiler Option COMPAT74 = NO

An explanation of the file status codes is given in the next sections for COMIPAT74 = NO.

### Sequential Files

The Sequential file codes are described in the following sections.

#### 0 Successful Completion

- 00      No further information available.
- 04      READ statement successfully executed, but the length of the record does not conform to the fixed file attributes for the file.
- 05      OPEN statement successfully executed, but the referenced optional file is not present at the time the OPEN statement is executed. If the OPEN mode is I-O or EXTEND, the file has been created.
- 07      OPEN statement successfully executed. However, for a CLOSE statement with the NO REWIND, REEL/UNIT, or FOR REMOVAL phrase, or for an OPEN statement with the NO REWIND phrase, the referenced file is a non-reel/unit medium.

#### 1 AT END

- 10      A sequential READ was attempted, and no next logical record exists in the file because the end-of file has been reached, or the READ statement was attempted for the first time on an optional Input file that is not present.
- 11      End-of-volume has been reached for a tape file; user program indicates no automatic volume switch.

#### 3 Permanent Error Condition

- 30      A hardware error (for example, tape parity error) occurred. Refer to SVC CHECK in the *VS Operating System Services Reference*. This value is not returned for program-related errors.
- 34      Attempt was made to WRITE beyond the externally defined boundaries of a file.
- 35      OPEN statement with the INPUT, I-O, or EXTEND phrase was attempted on a nonoptional file that is not present.
- 37      An OPEN statement was attempted on a file that does not support the open mode specified. The error occurred because (1) the EXTEND or OUTPUT phrase is

specified for a file that will not support WRITE operations, (2) The I-O phrase is specified for a file that will not support both input and output operations, and (3) the INPUT phrase is specified, but the file will not support read operations.

- 38 An attempt was made to open a file that has been closed with LOCK.
- 39 Returned by OPEN statement. A conflict was detected between the fixed file attributes and the attributes specified for the file in the program. For example, an unacceptable tape label or density was specified for the file, there were insufficient user access rights, or the file format was different from that specified

The fixed file attributes are

- File ORGANIZATION
- Record length
- COMPRESSED or not

#### **4 Logic Error**

- 41 File possession conflict; an OPEN statement was attempted for a file already open.
- 42 A CLOSE statement was attempted for a file not open
- 43 For a mass storage file in sequential ACCESS mode, the last I-O statement executed for the associated file prior to the execution of a REWRITE statement was not a successfully executed READ statement.
- 44 Boundary violation. An attempt was made to write or rewrite a record larger than the largest or smaller than the smallest record allowed by the RECORD IS VARYING clause of the associated file-name; or, an attempt was made to rewrite a record to a Sequential file, and the record is not the same size as the record being replaced.
- 46 A sequential READ statement was attempted on a file open in the INPUT or I-O mode, and no valid next record has been established because (1) the preceding READ statement was unsuccessful but did not cause an AT END condition, or (2) the preceding READ statement caused an AT END condition.
- 47 A READ statement was attempted on a file not open in INPUT or I-O mode.
- 48 A WRITE statement was attempted on a file not open in the OUTPUT or EXTEND mode.
- 49 A REWRITE statement was attempted on a file not open in I-O mode.



## **6 I-O Cancel Condition**

- 60 No further information available.

## **7 Timeout**

- 70 No further information available.

## **8 SHARED Mode Error**

- 80 No further information available.
- 82 Unsuccessful attempt by the system to update a file label.
- 83 SHARED mode malfunction that can be corrected only by doing an Initial Program Load (IPL). Refer to the *VS System Operator's Reference*.
- 84 Attempt to REWRITE a variable-length record whose record length is greater than the maximum record size specified for the file.
- 85 Attempt to update a file for which the user has Read-only access.
- 86 Invalid sequence of function requests. For example, an attempt to do a START HOLD on a file while another file is already held.

## **9 Special Conditions**

- 95 Invalid function request or invalid sequence of function requests for a given combination of device type, OPEN mode, and file organization.
- 97 An attempt to execute an I-O operation for a record of invalid length.

## **Indexed Files**

The Indexed file codes are described in the following sections.

### **0 Successful Completion**

- 00 No further information available.
- 02 Duplicate key detected. For a READ statement, the key value for the current key of reference is equal to the value of the same key in the next record within the current key of reference.
- 04 READ statement successfully executed, but the length of the record does not conform to the fixed file attributes for the file.

- 05 OPEN statement successfully executed, but the referenced optional file is not present at the time the OPEN statement is executed. If the OPEN mode is I-O or EXTEND, the file has been created.

## **1 AT END**

- 10 A sequential READ was attempted, and no next logical record exists in the file because the end-of-file has been reached, or the READ statement was attempted for the first time on an optional input file that is not present.

## **2 INVALID KEY**

- 21 Sequence error for a sequentially accessed Indexed file. For a WRITE to a file open in OUTPUT or EXTEND mode, the value of the prime record key is not greater than any record existing in the file; or the record key value has been changed by the COBOL program between the successful execution of a READ statement and the execution of the next REWRITE statement for the file.
- 22 Duplicate record key value. An attempt was made to WRITE or REWRITE a record that would create a duplicate prime record key or a duplicate alternate record key for which duplicates are not allowed.
- 23 No record found. An attempt was made to access a record that does not exist, a START or RANDOM READ statement was attempted on an optional Input file that is not present, or an attempt was made to DELETE a record that does not exist.
- 24 Boundary violation. An attempt was made to WRITE beyond the externally defined boundaries of the file.

## **3 Permanent Error Condition**

- 30 A hardware error occurred. This value is not returned for program-related errors.
- 35 OPEN statement with the INPUT, I-O, or EXTEND phrase was attempted on a nonoptional file that is not present.
- 37 An OPEN statement was attempted on a file that does not support the open mode specified. This error occurred because (1) the EXTEND or OUTPUT phrase is specified for a file that will not support WRITE operations, (2) the I-O phrase is specified for a file that will not support both input and output operations, and (3) the INPUT phrase is specified but the file will not support read operations.
- 38 An attempt was made to open a file that has been closed with LOCK.
- 39 Returned by OPEN statement. A conflict was detected between the fixed file attributes and the attributes specified for the file in the program. For

example, an unacceptable tape label or density was specified for the file, there were insufficient user access rights, or the file format was different from that specified

The fixed file attributes are

- File ORGANIZATION
- Record length
- COMPRESSED or not
- Position and length of primary or alternate keys
- Alternate key does or does not allow duplicates

#### **4 Logic Error**

- 41 File possession conflict; an OPEN statement was attempted for a file already open.
- 42 A CLOSE statement was attempted for a file not open.
- 43 For a mass storage file in sequential ACCESS mode, the last I-O statement executed for the associated file prior to the execution of a REWRITE or DELETE statement was not a successfully executed READ statement.
- 44 Boundary violation; an attempt was made to write or rewrite a record larger than the largest or smaller than the smallest record allowed by the RECORD IS VARYING clause of the associated file-name.
- 46 A sequential READ statement was attempted on a file open in the INPUT or I-O mode. No valid next record has been established because (1) the preceding START statement was unsuccessful, (2) the preceding READ statement was unsuccessful but did not cause an AT END condition, or (3) the preceding READ statement caused an AT END condition
- 47 A READ statement was attempted on a file not open in the INPUT or I-O mode.
- 48 A WRITE statement was attempted on a file not open in the OUTPUT or EXTEND mode.
- 49 A REWRITE or DELETE statement was attempted on a file not open in the I-O mode.

#### **6 I-O Cancel Condition**

- 60 No further information available.

#### **7 Timeout**

- 70 No further information available.

## **8 SHARED Mode Error**

- 80 No further information available.
- 82 Unsuccessful attempt by the system to update a file label.
- 83 SHARED mode malfunction that can be corrected only by doing an Initial Program Load (IPL).
- 84 Attempt to REWRITE a variable-length record whose record length is greater than the maximum record size specified for the file.
- 85 Attempt to update a file for which the user has Read-only access.
- 86 Invalid sequence of function requests. For example, an attempt to do a START HOLD on a file while another file is already held

## **9 Special Conditions**

- 95 Invalid function request or invalid sequence of function requests for a given combination of device type, OPEN mode, and file organization.
- 97 An attempt to execute an I-O operation for a record of invalid length.
- 98 Use of nonexistent alternate record key when attempting to REWRITE or WRITE a record of an alternate-indexed file.

## **Relative Files**

The Relative file status codes are described in the following sections.

### **0 Successful Completion**

- 00 No further information available.
- 04 READ statement successfully executed, but the length of the record does not conform to the fixed file attributes for the file.
- 05 OPEN statement successfully executed, but the referenced optional file is not present at the time the OPEN statement is executed. If the OPEN mode is I-O or EXTEND, the file has been created.

### **1 AT END**

- 10 A sequential READ was attempted. No next logical record exists in the file because the end-of-file has been reached, or the READ statement was attempted for the first time on an optional input file that is not present.

## 2 INVALID KEY

- 22 Duplicate record key value. An attempt was made to WRITE or REWRITE a record that would create a duplicate record key.
- 23 No record found. An attempt was made to READ, REWRITE, or START on a record that does not exist; or a START or random READ statement was attempted on an optional input file that is not present; or an attempt was made to DELETE a record that does not exist.
- 24 Boundary violation. An attempt was made to WRITE beyond the externally defined boundaries of the file; a sequential WRITE statement was attempted; and the number of significant digits in the relative record number is larger than the size of the relative key data item described for the file.

## 3 Permanent Error Condition

- 30 A hardware error (for example, parity error) occurred. Refer to SVC CHECK in the *VS Operating System Services Reference*. This value is not returned for program-related errors.
- 35 OPEN statement with the INPUT, I-O, or EXTEND phrase was attempted on a nonoptional file that is not present.
- 37 An OPEN statement was attempted on a file that does not support the OPEN mode specified. The error occurred because (1) the EXTEND or OUTPUT phrase is specified for a file that will not support WRITE operations; (2) The I-O phrase is specified for a file that will not support both input and output operations; and (3) the INPUT phrase is specified, but the file will not support read operations.
- 38 An attempt was made to open a file that has been closed with LOCK.
- 39 Returned by OPEN statement. A conflict was detected between the fixed file attributes and the attributes specified for the file in the program. For example, an unacceptable tape label or density was specified for the file, there were insufficient user access rights, or the file format was different from that specified.

The fixed file attributes are

- File ORGANIZATION
- Record length

#### **4 Logic Error**

- 41 File possession conflict; an OPEN statement was attempted for a file already open.
- 42 A CLOSE statement was attempted for a file not open.
- 43 For a mass storage file in sequential ACCESS mode, the last I-O statement executed for the associated file prior to the execution of a REWRITE or DELETE statement was not a successfully executed READ statement.
- 44 Boundary violation. Either (1) an attempt was made to write or rewrite a record larger than the largest or smaller than the smallest record allowed by the RECORD IS VARYING clause of the associated file-name, or (2) an attempt was made to rewrite a record that is not the same size as the record being replaced.
- 46 A sequential READ statement was attempted on a file open in the INPUT or I-O mode, and no valid next record has been established because (1) the preceding START statement was unsuccessful, (2) the preceding READ statement was unsuccessful but did not cause an AT END condition, or (3) the preceding READ statement caused an AT END condition.
- 47 A READ statement was attempted on a file not open in INPUT or I-O mode.
- 48 A WRITE statement was attempted on a file not open in the OUTPUT or EXTEND mode.
- 49 A REWRITE or DELETE statement was attempted on a file not open in I-O mode.

#### **6 I-O Cancel Condition**

- 60 Insufficient space for file of specified size to be opened in OUTPUT mode.

#### **7 Timeout**

- 70 No further information is available.

#### **9 Special Conditions**

- 97 An attempt to execute an I-O operation for a record of invalid length.

## Appendix F Printer Control Characters

This appendix describes the characters used to write a record to a printer file. The 2-byte printer control characters are defined in the FIGURATIVE-CONSTANTS paragraph in the Environment Division of the program. The format of the WRITE statement for printer files using printer control characters is

WRITE print-record AFTER ADVANCING user-figurative-constant.

The user-figurative-constant is formed by turning on the bits corresponding to the desired function. The following table describes the bits used in the 2-byte printer control area.

**Printer Control Characters**

Control		
Byte	Bit	Function
0	0	Line or channel spacing select  0 = Space number of lines specified in the second control byte. 1 = Skip to the channel specified in the second control byte.
	1	0 = Space before printing. 1 = Space after printing. 0 = Normal width characters.
	2	1 = Double width (expanded print) characters.
	3	1 = Actuate hardware alarm.
	4 - 7	RESERVED.
1	0	RESERVED.
	1 - 7	Binary number of lines to space (0 - 127) or channel for skip (1 - 12).

Features such as the printer hardware alarm and channel skipping are not available on all printers; before coding the figurative constant for the feature, be sure it is supported on the printer.

The following table illustrates the figurative constant settings for specified actions on a WRITE statement for printer files.

**Figurative Constant Settings for Printer Control**

<b>Figurative Constant</b>	<b>Function</b>
0004	Space 4 lines before printing.
4004	Space 4 lines after printing.
8001	Skip to Channel 1 on the printer (if the printer supports channel skipping).
2003	Space 3 lines before printing. If the printer supports expanded print, print the line by using expanded print characters; if the printer does not support expanded print, the expanded print bit will be ignored.
707F	(Assume the printer supports all features.) <ol style="list-style-type: none"><li>1. Print the line by using expanded print characters.</li><li>2. Actuate the hardware alarm.</li><li>3. Space 127 lines after printing.</li></ol>



# Appendix G Intermediate Results

## Introduction

This appendix specifies when COBOL 85 uses intermediate results in arithmetic computations and describes the algorithm used to determine the characteristics of the intermediate result.

A series of arithmetic operations is treated as a succession of operations performed according to the rules of evaluation. The result of each successive operation is defined to be an intermediate result. Intermediate results are used in arithmetic expressions.

## Usage Types of Intermediate Results

The usage type of an intermediate result can be native BINARY, COBOL BINARY, COMPUTATIONAL, or floating point and is determined as follows:

- If the operands are combined with the exponentiation operator, the intermediate result is always floating point. If any of the operands are floating point functions, the intermediate result is always floating point.
- If the operands are compared, or are combined using the addition, subtraction, multiplication, or division operator, the intermediate result is determined as follows:
  - If the usage types of the operands are the same, the intermediate result is also of that type.
  - If the usage types of the operands differ, the type of the intermediate result is the one of the two types that is higher in the hierarchy of types. The hierarchy of types from low to high is
    1. Native BINARY
    2. COBOL BINARY
    3. COMPUTATIONAL
    4. Floating point

## Precision of Intermediate Results

An intermediate result has an implied PICTURE that is defined as follows:

1. For operands of any type combined with the exponentiation operator, if the operands of the exponentiation operator are not COMPUTATIONAL, they are first converted to COMPUTATIONAL and then to floating point. Calculations are performed in floating decimal on the converted operands. The intermediate result, which is in floating point, contains at least the 14 most significant decimal digits. If assignment of the intermediate result is required, it is converted to COMPUTATIONAL before assignment.

Therefore, whenever exponentiation and other operations are mixed in an arithmetic expression, it is best to have the exponentiation evaluated last or as close to the end as possible.

2. For expressions in which both operands are native BINARY, calculations (except exponentiation) on expressions using only native BINARY data are done in native BINARY. Intermediate results have up to 31 bits of precision. When a BINARY data item is combined with an item of a different type using the addition, subtraction, multiplication, or division operator, the BINARY data item is converted to the other data type.
3. For all other expressions, precision of intermediate results in all calculations not covered in the previous two categories are determined as follows. Let OP1 and OP2 be the operands of the expression.

ROPI - Indicates the number of digits to the right of the decimal point in OP 1.

LOPI - Indicates the number of digits to the left of the decimal point in OP 1.

ROP2 - Indicates the number of digits to the right of the decimal point in OP2.

LOP2 - Indicates the number of digits to the left of the decimal point in OP2.

RI- Indicates the number of digits to the right of the decimal point in *the* intermediate result.

LI - Indicates the number of digits to the left of the decimal point in the intermediate result.

RF-Indicates the number of digits to the right of the decimal point in the final result field.

LF -Indicates the number of digits to the left of the decimal point in the final result field.

For addition (OP 1 + OP2) or subtraction (OP 1 - OP2)

RI = max (ROP1, ROP2)

LI = max (LOP I, LOP2) + 1

Form ultiplication (OP I \* OP2)

RI = ROPI + ROP2

LI = LOP I + LOP2

For division (OP 1 / OP2) the quotient

RI = max (ROPI, ROP2, RF), plus one if ROUNDED is specified for the final result

LI = LOP I + ROP2

The remainder

RI = ROP2 + (RI of the quotient)

LI = LOP2 - (RI of the quotient)

For comparison (OP I compared with OP2)

RI = max (ROPI, ROP2)

LI = max (LOP1, LOP2)

In any instance, if RI + LI exceeds 30, truncation of all but the 30 least significant digits will occur.

The implied PICTURE of the intermediate result is S9(LI)V9(RI), where LI + RI may exceed 18, but will not exceed 30.

In making precision computations by using numeric literals, intermediate calculations are made as if the literal had been defined with the minimum signed PICTURE describing the literal. For example, 123.4 and 00123.400 are both considered as PIC S999V9. This is equivalent to removing non-significant zeros.

It is the programme's responsibility to ensure that the operands of an arithmetic expression are defined to give the desired accuracy in the result.

# Appendix H VS Extension Rights

## Introduction

This appendix describes the use of VS extension rights. Extension rights are features of DMS Sharing that allow a program to request the exclusive right to request resources incrementally, claiming them as needed. Like DMS/TX, the use of extension rights allows multiple users simultaneous access to data. DMS/TX, however, offers a more complete solution to data consistency and concurrency.

Under DMS Sharing, programs using the standard HOLD statement request all the needed resources at the same time. Before additional resources are requested any resources already held must be released. Extension rights allow a program to hold additional resources without releasing resources already held. However, a program cannot acquire extension rights if it is currently holding resources. In order to request extension rights, a program must first release, by means of the FREE ALL statement, any resources it may be holding.

A program requests extension rights with the HOLD EXTENSION-RIGHTS statement. Only one program can hold extension rights at a time. You can use the TIMEOUT and HOLDER-ID phrases to cause the program to wait for the request to be granted and to return the user-ID of the current extension rights holder if the request cannot be granted in the specified time.

Once extension rights are obtained, the needed resources are requested by the standard HOLD and HOLD LIST statements. It is not necessary to release currently held resources before requesting new resources. The resources held by the program remain held until they are released together by FREE ALL. Refer to the section titled "FREE EXTENSION-RIGHTS Statement."

Extension rights do not guarantee that the requested resources are not already held by another program, nor do extension rights held by one program prevent other programs from requesting and obtaining resources. When a program obtains extension rights, any resource currently held by another program remains held. Another program can request resources if it is not already holding resources, but it cannot request additional resources unless it releases the resources it already has. The restriction that only one program at a time can have extension rights is imposed to avoid the possibility of deadlock, the condition in which two programs cannot proceed because each is holding resources needed by the other.

HOLD EXTENSION-RIGHTS should be used with extreme care. Extension rights should be invoked only if the program absolutely requires the claim-as-you-go strategy; that is, if it cannot be determined beforehand what resources will be needed and if none of the resources can be released until all of the held resources have been processed. For example, a customer may be ordering several different items. When updating the order file, it is not known beforehand which inventory records must be updated; yet all the inventory records should be updated before the order processing is completed. Extension rights allow the program to request selected records of the inventory file as needed.

After all needed resources have been identified and held, extension rights should be released as soon as possible, so that another program can obtain extension rights. Extension rights are released by the FREE statement.

To prevent program cancellation in the event of an unsuccessful execution, use the ON ERROR clause with the FREE EXTENSION-RIGHTS statement. The ON ERROR clause causes the accompanying imperative statement to be executed and the special register RETURN-CODE to contain the return code.

The following section contains a complete COBOL 85 program illustrating the COBOL statements that support the use of extension rights. In this example, it is not known beforehand which records of EMPLOYEE-FILE are needed, since the value of DEPARTMENT is entered at the workstation. The extension rights allow the program to hold PERSONNEL-FILE and then hold whatever records of EMPLOYEE-FILE are needed, without first releasing PERSONNEL-FILE.

Lines 40 to 45 illustrate the HOLD EXTENSION-RIGHTS statement, with TIMEOUT and HOLDER-ID phrases. If extension rights are held by another program, this program waits 5 seconds (specified in the TIMEOUT phrase). If the extension rights are not released within 5 seconds, the data item WHO-HAS-IT contains the ID of the user running the program that holds the extension rights, and a message is displayed

### **COBOL 85 Program - HOLD EXTENSION-RIGHTS**

```
000001 IDENTIFICATION DIVISION. 000002 PROGRAM-ID. EXRIGHTS. 000003 ENVIRONMENT
DIVISION. 000004 INPUT-OUTPUT SECTION. 000005 FILE-CONTROL.
000006 SELECT EMPLOYEE-FILE
000007     ASSIGN TO "EMPLOYEE",          "DISK",          NODISPLAY,
000008 ORGANIZATION IS INDEXED
000009 ACCESS MODE
000010 IS DYNAMIC
000011     RECORD KEY IS EMPLOYEE-NUMBER.
000012 SELECT PERSONNEL-FILE
000013 ASSIGN TO "PERSONS",          "DISK",          NODISPLAY,
000014 ORGANIZATION IS INDEXED
000015 ACCESS MODE IS DYNAMIC
000016 RECORD KEY IS PERSONNEL-RECORD-NUMBER.
000017 DATA DIVISION.
000018 FILE SECTION.
000019 FD EMPLOYEE-FILE
000020 LABEL RECORDS ARE STANDARD.
000021 01 EMPLOYEE-RECORD.
000022 03 EMPLOYEE-NUMBER.
000023 05 DEPARTMENT PIC 9(3).
000024 05 FILLER PIC 99.
000025 03 EMPLOYEE-NAME PIC X(20).
000026 FD PERSONNEL-FILE
000027 LABEL RECORDS ARE STANDARD.
000028 01 PERSONNEL-RECORD.
000029 03 PERSONNEL-RECORD-NUMBER PIC 9(5).
000030 03 PERSONNEL-DATA PIC X(20).
000031 WORKING-STORAGE SECTION.
000032 77 WHO-HAS-IT PIC X(3).
000033 PROCEDURE DIVISION.
000034 START-PROGRAM.
000035 PERFORM HOLD-RIGHTS THRU END-HOLD.
000036 STOP RUN.
000037 HOLD-RIGHTS.
000038     OPEN SHARED EMPLOYEE-FILE.
000039     OPEN SHARED PERSONNEL-FILE.
```

```

000040      HOLD EXTENSION-RIGHTS
000041                      TIMEOUT OF 5 SECONDS
000042                      HOLDER-ID IN WHO-HAS-IT
000043      DISPLAY WHO-HAS-IT
000044      " is holding EXTENSION-RIGHTS."
000045      GO TO HOLD-RECORDS.
000046      DISPLAY "EXTENSION-RIGHTS are held by this program.".
000047 HOLD-RECORDS.
000048 HOLD RECORDS OF PERSONNEL-FILE FOR RETRIEVAL
000049                      TIMEOUT OF 5 SECONDS
000050 HOLDER-ID IN WHO-HAS-IT
000051 DISPLAY WHO-HAS-IT
000052      " is holding PERSONNEL-FILE.'
000053 GO TO CLOSE-FILES.
000054 DISPLAY "PERSONNEL-FILE is held by this program.".
000055 ACCEPT DEPARTMENT.
000056 HOLD RECORDS OF EMPLOYEE-FILE FOR UPDATE
000057 WITH KEYS INITIAL 3 CHARACTERS OF EMPLOYEE-NUMBER
000058      TIMEOUT OF 5 SECONDS
000059      HOLDER-ID IN WHO-HAS-IT
000060 DISPLAY WHO-HAS-IT
000061 " is holding records with first 3 characters of
000062-      "DEPARTMENT in EMPLOYEE-FILE."
000063 GO TO CLOSE-FILES.
000064 DISPLAY "Records with first 3 characters of "DEPARTMENT
000065-      " in EMPLOYEE-FILE are held by this program.".
000066 RELEASE-RIGHTS.
000067 FREE EXTENSION-RIGHTS.
000068 DISPLAY "The EXTENSION-RIGHTS are released, but the
000069-      " resources are still held.".
000070 RELEASE-RESOURCES.
000071 FREE ALL.
000072 DISPLAY "PERSONNEL-FILE and EMPLOYEE-FILE are no longer
000073-      " held by this program, but both files are still open.".
000074 CLOSE-FILES.
000075 CLOSE EMPLOYEE-FILE, PERSONNEL-FILE.

000076 END-HOLD.
000077 EXIT.

```

## FREE EXTENSION-RIGHTS Statement

A program should release resources and/or extension rights when the need for them has been satisfied. This is done by coding the FREE statement. If the program does not code the FREE statement, other programs are prevented from obtaining needed resources. The FREE statement can either free all resources including extension rights (FREE ALL) or free extension rights only (FREE EXTENSION-RIGHTS). Thus, by coding FREE EXTENSION-RIGHTS as soon as the needed resources have been held, a program can continue to hold the resources for whatever processing might be necessary without preventing another program from obtaining the extension rights.

### HOLD EXTENSION-RIGHTS Format

```
HOLD EXTENSION-RIGHTS

      integer-1      SECOND
TIMEOUT OF
      data-name-1    SECONDS

      (HOLDER-ID IN data-name-2)

      imperative-statement-1
      NEXT SENTENCE

[ END-HOLD ]
```

#### *General Rules*

1. HOLD EXTENSION-RIGHTS allows a program the exclusive right to hold more than one file and/or a range of records of an Indexed file, without implicitly releasing resources already held. No resources are actually held by HOLD EXTENSION-RIGHTS.
2. Only one program can have extension-rights at a time. It is recommended that as soon as the resources have been held, a FREE EXTENSIONRIGHTS be issued so that another program can obtain extension-rights.
3. If the TIMEOUT phrase is specified, and the HOLD EXTENSIONRIGHTS cannot be completed in data-name-1 or integer-1 seconds, then imperative-statement-1 is executed. If the number of seconds specified is zero, the timeout exit will immediately be taken if the HOLD EXTENSION-RIGHTS cannot be completed. If the HOLDER-ID phrase is specified in the TIMEOUT phrase, the user ID of the user currently holding the extension-rights is moved to data-name-2. Data-name-2 must be defined in the Working-Storage section or Linkage section and have a PICTURE of X(3).
4. If the TIMEOUT phrase is not specified and the HOLD EXTENSIONRIGHTS cannot be satisfied, the program waits until the user holding the extension-rights frees them.
5. If there is no TIMEOUT phrase, the user's program will be canceled upon unsuccessful execution of the statement.

## **FREE EXTENSION-RIGHTS Format**

```
      ALL
FREE
      EXTENSION RIGHTS

      [ ON .ERROR imperative-statement-1 ]

      [ NOT ON ERROR imperative-statement-2 ]

      [ END-FREE ]
```

### ***General Rules***

1. FREE ALL releases from hold status all resources (including extension-rights) held by the program.
2. FREE EXTENSION-RIGHTS removes extension-rights from the user but does not release any resources held.
3. If there is an ON ERROR clause, any unsuccessful execution of this statement (with a nonzero return code) will cause the ERROR imperative-statement to be executed. In this situation, the special register RETURN-CODE contains the return code of the statement being executed.

## Appendix I Wang COBOL 85, ANSI, and FIPS Standards

The 1985 American National Standards Institute (ANSI) specifications for COBOL (ANSI X3.23-1985) are divided into 11 modules organized according to processing functions. These modules define different features of the language. The modules are Nucleus, Sequential I-O, Relative I-O, Indexed I-O, InterProgram Communication, Sort-Merge, Source Text Manipulation, Report Writer, Communication, Debug, and Segmentation.

All the modules except Report Writer and Sort-Merge contain a higher and lower level. In each case, the features of the lower level are supported in the higher level, except that the higher level can remove some restrictions that are enforced in the lower level.

The standard was amended in 1989 with the addition of the Intrinsic Functions module, as defined in ANSI XC3.23a-1989.

The Federal Information Processing Standard (FIPS) for COBOL 85 (FIPS PUB 21-2) is based on the ANSI standard. FIPS divides the modules of ANSI COBOL 85 into three subsets and four optional modules. The three subsets are Minimum, Intermediate, and High; the four optional modules are Report Writer, Communications, Debug, and Segmentation. The four optional modules are not an integral part of any of the subsets; but none, or all, or any combination of the optional modules can be included in any subset.

The High subset is composed of all language elements of the highest level of all required modules. The Intermediate subset is composed of all language elements of level 1 of all required modules. The Minimum subset is composed of all language elements of level 1 of the Nucleus, Sequential I-O, and Inter-Program Communication modules.

Each FIPS level is composed of features from the high or low levels of ANSI modules. The numbers in the following table refer to the levels of the corresponding ANSI module. The digit 1 represents the low level; 2 designates the high level. A dash means that the corresponding module is omitted from that FIPS level.



### Federal Information Processing Standard

Required	Minimum	Intermediate	High
Nucleus	1	1	2
Sequential I-O	1	1	2
Relative I-O	-	1	2
Indexed I-O	-	1	2
Inter-Program Communication	1	1	2
Intrinsic Functions	-	-	1
Sort-Merge	-	1	1
Source Text Manipulation	-	1	2
<b>Optional</b>			
Report Writer	-, or 1	-, or 1	-, or 1
Communication	-, 1, or 2	-, 1, or 2	-, 1, or 2
Debug	-, 1, or 2	-, 1, or 2	-, 1, or 2
Segmentation	-, 1, or 2	-, 1, or 2	-, 1, or 2

The next tables list features of VS and ANSI COBOL 85, organized according to the source-program divisions. The letters in the tables have the following meanings:

- C indicates that the feature is accepted by the Compiler as a comment entry only.
- E indicates that the feature is a VS extension to the ANSI standard.
- I indicates that the feature is an implementor name.
- N indicates that the feature is not supported. Note, however, that all ANSI reserved words are recognized as such by the VS Compiler and cannot be user-defined words.
- Y indicates that the feature is supported.

The column entitled "ANSI" indicates whether a feature conforms to the ANSI Standard. A dash in the column indicates that the feature is absent from the Standard. A 3-character abbreviation indicates the module to which a feature belongs. The number preceding the abbreviation indicates the ANSI level to which the feature belongs: 1 indicates the lower level; 2 indicates the higher level. Since the Segmentation, Report Writer, Debugging, and Communication modules are not supported at **all**, they are not included. The meanings of the abbreviations are as follows:

Abbreviation	Meaning
NUC	Nucleus
SEQ	Sequential I-O
REL	Relative I-O
INX	Indexed I-O

### Summary of Differences in Language Concepts

ELEMENT	ANSI	Wang
<b>Character Set</b>		
Single-character substitution allowed	1 NUC	N
Double-character substitution allowed	1 NUC	N
<b>Character Strings</b>		
User-defined words		
Routine-name	1 NUC	C
User-figurative-constant	--	E
<b>System-names</b>		
Computer-name	1 NUC	C
Language-name	1 NUC	C
<b>Uniqueness of Reference</b>		
FAC qualifier	--	E
ORDER AREA qualifier	--	E
<b>Reference Format</b>		
Comment entries		
Percent sign (%) comment line	--	E
<b>Source Text Manipulation</b>		
COPY statement		
In library text (Nested COPY)	--	E

# Summary of Differences in Environment Division

ELEMENT	ANSI	Wang
<b>Configuration Section</b>		
SOURCE-COMPUTER paragraph		
MEMORY-SIZE clause	1 NUC	C
SPECIAL-NAMES paragraph		
SWITCH-1,..., SWITCH-7	--	I
FIGURATIVE-CONSTANTS paragraph	--	E
<b>Input-Output Section</b>		
FILE-CONTROL paragraph		
File control entry		
ACCESS MODE clause		
RANDOM for Sequential organization	--	E
DYNAMIC for Sequential organization	--	E
ALTERNATE RECORD KEY clause		
Integer option	--	E
ASSIGN clause		
DISK, STANDARD-DISK, DISPLAY,		
PRINTER, or TAPE	--	I
NODISPLAY phrase	--	E
CURSOR clause	--	E
BUFFER clause	--	E
PFKEY clause	--	E
PADDING CHARACTER clause	2 SEQ	C
I-O CONTROL paragraph		
MULTIPLE FILE TAPE clause	2 SEQ	C
RERUN clause	1 SEQ	C
SAME AREA clause	1 SEQ	C
	1 REL	C
	1 INX	Y

### Summary of Differences in Data Division

ELEMENT	ANSI	Wang
<b>File section</b>		
File description entry		
CODE-SET clause	1 SEQ	C
DATA RECORDS clause	1 SEQ	C
	1 REL	C
	1 INX	C
RECORD clause		
COMPRESSED phase	--	E
VALUE OF clause		
Implementor-name series		
DATABASE-NAME	--	E
RECOVERY-BLOCKS	--	E
RECOVERY-STATUS	--	E
<b>Working-Storage section</b>		
Record description entry		
COLUMN clause	--	E
LINE clause	--	E
OBJECT clause	--	E
RANGE clause	--	E
ROW clause	--	E
SOURCE clause	--	E
USAGE clause		
DISPLAY-WS	--	E
VALUE clause		
User-figurative-constant	--	E

# Summary of Differences in Procedure Division

ELEMENT	ANSI	Wang
<b>Conditional expressions</b>		
Mask test	--	E
Modified Data Tag condition	--	E
<b>Arithmetic statements</b>		
Composite of operands limited to 30 digits	--	E
<b>ACCEPT statement</b>		
FROM mnemonic-name phrase	2 NUC	C
<b>DISPLAY statement</b>		
UPON mnemonic-name phrase	1 NUC	C
<b>DISPLAY AND READ statement</b>	--	E
<b>FREE statement</b>	--	E
<b>HOLD statement</b>	--	E
<b>MOVE statement</b>		
WITH CONVERSION phrase	--	E
<b>OPEN statement</b>		
NO REWIND phrase	2 SEQ	C
REVERSED phrase	2 SEQ	C
SHARED phrase and SHARED series	--	E
<b>READ statement</b>		
ALTERED phrase	--	E
HOLD phrase	--	E
MODIFIABLE phrase	--	E
TIMEOUT phrase	--	E

(continued)

### Summary of Differences in Procedure Division

ELEMENT	ANSI	Wang
<b>REWRITE Statement</b> AFTER Phrase	--	E
<b>ROLLBACK Statement</b>	--	E
<b>START Statement</b> Use with ORGANIZATION IS SEQUEN- TIAL	--	E
<b>USE Statement</b> FUNCTION	--	E
AFTER DEADLOCK	--	E
<b>WRITE Statement</b> TIMEOUT phrase	--	E

## Appendix J Obsolete Elements in COBOL 85

The ANSI X3.23-1985 standard has categorized the following COBOL elements as obsolete in COBOL 85:

1. ALL literal associated with a numeric or numeric edited item. The figurative constant, ALL literal, is obsolete when the literal is a numeric or numeric edited item and when the length of the literal is greater than one.

Justification: The results of moving an ALL literal to a numeric data item are often unexpected. For example, according to the interpretation contained in X3J4 interpretation document B-23, the statements

```
01      A      PIC 99V99
```

```
MOVE ALL "99" TO A.  
MOVE ALL "123" TO A.
```

give values of 99.00 and 31.00, respectively. (The latter move is equivalent to MOVE "1231" to A. Decimal point alignment causes the "12" to be truncated.)

2. AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, and SECURITY paragraphs (1 NUC).

Justification: The purpose of these paragraphs can be achieved through the use of comment lines within the Identification Division.

The goal of cleaning up and regularizing the COBOL language has been achieved by declaring many implementor-defined elements obsolete. The format of the DATE-COMPILED and SECURITY paragraphs are examples of comment-entry paragraphs that are defined by the implementor.

The interaction of the COPY statement with the comment-entries in the AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, and SECURITY paragraphs is often ambiguous, that is, the presence of the word COPY in a comment-entry versus the use of the COPY statement in a comment-entry.

3. MEMORY SIZE clause (1 NUC).

Justification: This anachronistic feature of the language is a carry-over from the time when many systems required a specification of memory size allocation to load the nun unit. Memory capacity for a family of mainframe models often ranged from 8K to 64K maximum. COBOL programs used the MEMORY SIZE clause to generate objects for specific models.

This feature is considered to be a function more appropriately controlled by the host operating system in today's computing environment. In second Standard COBOL, the MEMORY SIZE clause was optional. Thus, there are no standard conforming COBOL implementations that require the use of the MEMORY SIZE clause to specify the object computer memory size.

4. RERUN clause of the I-O-CONTROL paragraph (1 SEQ, 1 REL, 1 INX).

Justification: Seven forms of the RERUN clause are provided. The implementor is required to support at least one form of the RERUN clause.

This feature is considered to be a function more appropriately controlled by the host operating system in today's computing environment.

The RERUN clause provides only one-half of a complete rerun/restart facility. That is, the syntax and **semantics for restart** are not specified. Due to the variety in forms of the RERUN clause, there is no guarantee that a program using this clause would be transportable.

5. MULTIPLE FILE TAPE clause in the I-O-CONTROL paragraph of the Environment Division (2 SEQ, 1 RPW).

Justification: The MULTIPLE FILE TAPE clause should be a function of the operating system and not the individual COBOL program.

6. LABEL RECORDS clause in the file description entry (1 SEQ, 1 REL, 1 INX, 1 RPW).

Justification: Specifying the presence of file labels is considered a function of the operating system and does not belong in the COBOL program.

7. VALUE OF clause in the file description entry (1 SEQ, 1 REL, 1 INX, 1 RPW).

Justification: Describing file label items is considered a function of the operating system and does not belong in the COBOL program.

8. DATA RECORDS clause of the file description entry (1 SEQ, 1 REL, 1 INX).

Justification: This clause is redundant and can cause misleading documentation.

9. ALTER statement (1 NUC).

Justification: Use of the ALTER statement results in a program that is difficult to understand and maintain. It provides no unique functionality since the GO TO DEPENDING statement can serve the same purpose.

10. ENTER statement (1 NUC).

Justification: The ENTER statement was a precursor of the CALL statement and of the calling of external programs. It provides no portability because it is optional and is defined by the implementor, it is therefore not a good candidate for standardization.

11. STOP literal statement (1 NUC).

Justification: If STOP literal-1 is specified, the execution of the run unit is suspended and literal-1 is communicated to the operator. Continuation of the execution of the run unit begins with the next executable statement when the implementor-defined procedure governing run unit reinitiation is instituted.

The function of the STOP literal statement is substantially defined by the implementor, and thus programs using it are not portable. The overlap in functionality allows the program to use a DISPLAY literal as a substitute.



## Appendix K Link Editing and the COBOL 85 Runtime

The COBOL 85 compiler generates a call to a runtime routine for the following language constructs:

WRKACCEP	ACCEPT from workstation
WRKACOS	Arccosine function
WRKASIN	Arcsine function
WRKATAN	Arctangent function
WRKCALID	CALL identifier
WRKCHR	Character at ordinal position in collating sequence function
WRKCOS	Cosine function
WRKCRDT	Current date function
WRKDANR	DISPLAY AND READ
WRKDEDIT	MOVE from numeric edited to numeric
WRKDSPLY	DISPLAY
WRKDTIT	Date of integer function
WRKDVPL	DIVIDE with operands longer than 15 digits
WRKDYIT	Day of integer function
WRKDYWK	ACCEPT DAY OF WEEK
WRKIDXC	INSPECT
WRKITDT	Integer of date function
WRKITDY	Integer of day function
WRKLFD	Math function support
WRKLOG	Natural logarithm function
WRKLOG 10	Base 10 logarithm function
WRKLOWR	Lowercase function
WRKMTPL	MULTIPLY with operands longer than 15 digits
WRKNMV	Numeric value function
WRKNMVC	Numeric value allowing currency symbol function
WRKORD	Ordinal position of character in collating sequence function
WRKPOWER	Exponentiation
WRKRAND	Pseudo random number generator function
WRKRVRS	Reverse character string function
WRKSBCK	Runtime checking of subscripts (if SUBCHK = YES)
WRKSIN	Sine function
WRKSOME	SORT or MERGE
WRKSQRT	Square root function
WRKSTCK	Runtime checking of reference modification subscripts
WRKSWITC	External switches
WRKTAN	Tangent function
WRKUPPR	Uppercase function

If a program contains any of these statements, the Linker must be run to include these routines in the object file before the object code can be executed.

If the Compiler option LINKLIB is not blank, the compiler runs the Linker to link runtime routines as part of the compilation process. If LINKLIB is blank, it does not. If a user intends to run the Linker to combine several compilation units into one run unit, it is better not to run the Linker as part of the compilation process. This way of proceeding has three advantages:

1. The sizes of the object files on disk are kept to a minimum, since copies of the runtime routines reside only in the combined object file and not in the individual object files.
2. The most recent copy of the runtime is always included.
3. The time taken to run the Linker during each compilation is saved.

The runtime routines all begin with the 3-letter prefix WRK. Further routines will be added to support additional language elements. In order to avoid possible name conflicts during execution of the Linker, do not begin program names with the 3-letter prefix WRK.