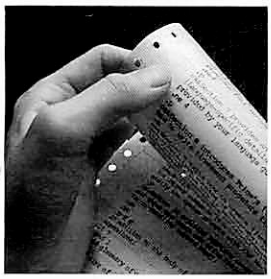
Prime Computer, Inc.

PRIME

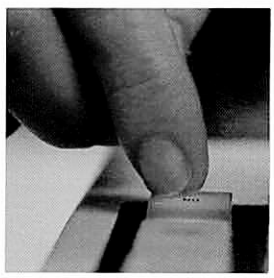
DOC5039-184L
Cobol 74
Reference Guide
Revision 18.4










COBOL 74 Reference Guide

DOC 5039-184

First Edition

by

Anne P. Ladd

This guide documents the software operation of the Prime Computer and its supporting systems and utilities as implemented at Master Disk Revision Level 18.4 (Rev. 18.4).

Prime Computer, Inc.
500 Old Connecticut Path
Framingham, Massachusetts 01701

COPYRIGHT INFORMATION

HOW TO ORDER TECHNICAL DOCUMENTS

U.S. Customers

Software Distribution
Prime Computer, Inc.
1 New York Ave.
Framingham, MA 01701
(617) 879-2960 X2053

Prime Employees

Communications Services
MS 15-13, Prime Park
Natick, MA 01760
(617) 655-8000 X4837

Customers Outside U.S.

Contact your local Prime subsidiary or distributor.

Prime INFORMATION

Contact your Prime INFORMATION dealer.

PRINTING HISTORY — COBOL 74 Reference Guide

Edition	Date	Number	Software Release
First Edition	May 1983	DOC5039-184	18.4

SUGGESTION BOX

All correspondence on suggested changes to this document should be directed to:

Anne P. Ladd
Technical Publications Department
Prime Computer, Inc.
500 Old Connecticut Path
Framingham, Massachusetts 01701

Contents

ABOUT THIS BOOK xi

1 OVERVIEW OF PRIME'S COBOL 74

Language Standards 1-1
Prime Extensions to the ANSI Standard 1-1
COBOL 74 Under PRIMOS 1-2
Program Environments 1-4
System Resources Supporting COBOL 74 1-5

2 COMPILING THE PROGRAM

Introduction 2-1
Using the Compiler 2-1
Compiler Options (Command Line Options) 2-2
COBOL Files -- Naming Conventions 2-14
Compiler Output 2-15

3 LOADING AND EXECUTING PROGRAMS

Loading Programs 3-1
Executing Loaded Programs -- Runtime 3-5
Switch Settings at Runtime 3-6
Runtime Error Messages 3-6

4 ELEMENTS OF PRIME COBOL 74

Divisions of a COBOL Program: A Summary 4-1
Format Notation 4-5
Coding Rules 4-7
Punctuation and Separators 4-8
The Prime COBOL 74 Character Set 4-9
Character-strings 4-11
Word Formation 4-12
Reserved Words 4-12
Programmer-defined Words 4-15
Literals 4-17
Data Levels 4-19
Classes and Categories of Data 4-20
Data Representation and Alignment 4-22
Algebraic Signs 4-31

Qualification, Subscripting, and Indexing 4-31
Arithmetic Expressions 4-35
Conditional Expressions 4-38

5 THE IDENTIFICATION DIVISION

IDENTIFICATION DIVISION 5-1
Example 5-4

6 THE ENVIRONMENT DIVISION

ENVIRONMENT DIVISION 6-1
OBJECT-COMPUTER 6-3
SPECIAL-NAMES 6-4
INPUT-OUTPUT SECTION 6-7
FILE-CONTROL 6-7
I-O-CONTROL 6-10
Example 6-12

7 THE DATA DIVISION

DATA DIVISION 7-1
FILE SECTION 7-3
File-description-entry 7-4
COMPRESSED/UNCOMPRESSED — Prime Extension 7-6
BLOCK CONTAINS 7-7
CODE-SET 7-8
DATA RECORDS 7-9
LABEL RECORDS 7-10
OWNER IS -- Prime Extension 7-11
RECORD CONTAINS 7-12
VALUE OF FILE-ID, VOL-ID, OWNER-ID 7-13
Record-description-entry 7-16
Level-number 7-18
Data-name or FILLER 7-22
BLANK WHEN ZERO 7-23
JUSTIFIED 7-24
OCCURS 7-25
PICTURE 7-26
REDEFINES 7-36
RENAMES 7-38
SIGN 7-40
SYNCHRONIZED 7-42
USAGE 7-43
VALUE 7-45
WORKING-STORAGE SECTION 7-48
LINKAGE SECTION 7-50
Example 7-52

8 THE PROCEDURE DIVISION

PROCEDURE DIVISION 8-1
Declarative Statements 8-7
Procedure Statements 8-7
Alphabetical List of All Verbs 8-8
Example 8-104

9 INTERPROGRAM COMMUNICATION

Function 9-1
LINKAGE SECTION 9-2
PROCEDURE DIVISION 9-4
CALL 9-5
ENTER 9-7
EXIT PROGRAM 9-8
GOBACK — Prime Extension 9-9
Loading and Executing
More Than One Program 9-10
Example 9-12

10 TABLE HANDLING

Definition 10-1
DATA DIVISION 10-2
OCCURS 10-2
USAGE 10-6
INDEXED BY 10-7
PROCEDURE DIVISION 10-8
SET 10-8
SEARCH 10-10
Strategy 10-16
Example 10-21

11 THE SORT-MERGE MODULE

Definition 11-1
Loading Sort and Merge Programs 11-2
ENVIRONMENT DIVISION 11-3
I-O-CONTROL 11-3
DATA DIVISION 11-5
FILE SECTION 11-5
Sort File Description 11-5
PROCEDURE DIVISION 11-6
MERGE 11-6
RELEASE 11-13
RETURN 11-15
SORT 11-17
Example 11-24

12 INDEXED SEQUENTIAL FILES

Function of the Indexed I-O Module 12-1
Loading and Executing Programs
With Indexed Files 12-1
Indexed File Concepts 12-2
Common Operations on Indexed Files 12-5
ENVIRONMENT DIVISION 12-7
INPUT-OUTPUT SECTION -- FILE CONTROL 12-7
I-O-CONTROL 12-10
DATA DIVISION 12-11
Record-description-entry 12-11
PROCEDURE DIVISION 12-12
CLOSE 12-12
DELETE 12-13
OPEN 12-14
READ 12-15
REWRITE 12-19
SEEK — Prime Extension 12-21
START 12-22
USE 12-25
WRITE 12-26
Example 12-28

13 RELATIVE FILES

Function of the Relative I-O Module 13-1
Loading and Executing Programs With
Relative Files 13-1
Relative File Concepts 13-2
Common Operations on Relative Files 13-5
ENVIRONMENT DIVISION 13-7
INPUT-OUTPUT SECTION -- FILE CONTROL 13-7
I-O-CONTROL 13-10
DATA DIVISION 13-11
Record-description-entry 13-11
RELATIVE KEY 13-11
PROCEDURE DIVISION 13-12
CLOSE 13-12
DELETE 13-13
OPEN 13-14
READ 13-16
REWRITE 13-19
SEEK — Prime Extension 13-21
START 13-22
USE 13-24
WRITE 13-25
Example 13-27

14 TAPE FILES

Introduction 14-1
Compiling, Loading, and
Executing Programs That Use Tape 14-2

File Assignments for Tape 14-4
Multivolume Tape Files 14-7
IDENTIFICATION DIVISION 14-8
ENVIRONMENT DIVISION 14-9
DATA DIVISION 14-11
BLOCK CONTAINS 14-12
CODE-SET 14-14
VALUE OF FILE-ID 14-15
PROCEDURE DIVISION 14-17
CLOSE 14-17
OPEN 14-18
READ 14-20
WRITE 14-21
Overview of the LABEL Command 14-22
Using LABEL 14-22
Errors Using LABEL 14-23
The HELP Facility 14-25
Example 14-26

APPENDIXES

A REFERENCE TABLES

COBOL Symbols A-2
COBOL Reserved Words A-6
ASCII Character Set and Collating Sequence A-10
EBCDIC Character Set and Collating Sequence A-15
File Status Codes A-17
Permissible I-O Statements A-20
Hexadecimal and Decimal Conversion A-21
Octal and Decimal Conversion A-22
Hexadecimal Addition Table A-22
Decimal Data Type (Overpunch Symbols) A-23

B ERROR MESSAGES

Compile Time Error Messages B-1
COBOL Runtime Error Messages B-2
MIDAS or MIDASPLUS Runtime Error Messages B-2
PRIMOS Error Messages B-3

C FIPS LEVELS C-1

D THE DEBUGGER INTERFACE

Overview D-1
Examples D-3

E CREATING INDEXED AND RELATIVE FILES: THE MIDASPLUS INTERFACE

Definitions E-1
CREATK E-2
KBUILD E-3
CREATK for Indexed Files E-3
KBUILD for Indexed Files E-8
CREATK for Relative Files E-11
KBUILD for Relative Files E-13

F COBOL 74 LIBRARY FILES F-1

G THE MAP OPTION

Example G-3

H THE XREF OPTION

Example H-2

I PRIME SUPPORT OF THE ANSI STANDARD

Prime Extensions to the ANSI Standard I-2

J IMPLEMENTATION-DEPENDENT FEATURES OF PRIME COBOL 74 IN REV. 18 J-1

K FILE ASSIGNMENTS WITH -OLD K-1

L CONVERSION INCOMPATIBILITIES WITH PRIME'S OLDER COBOL: REV. 18.4 AND HIGHER L-1

M GLOSSARY M-1

INDEX X-1

About This Book

PURPOSE AND AUDIENCE

The COBOL 74 Reference Guide (DOC5039) describes Prime COBOL 74 for software revision levels 18.4 and higher. The guide provides the necessary information for compiling, loading, executing, and debugging COBOL programs on a Prime system. It is designed to be used as a reference guide for an experienced COBOL programmer. Users unfamiliar with the language should read one of the many commercially available instruction books. Examples are:

Feingold, Carl. Fundamentals of Structured COBOL Programming. Dubuque: Wm. C. Brown Company, 1978.

McCracken, Daniel D. A Simplified Guide to Structured COBOL Programming. New York: John Wiley, 1976.

Philippakis, Andreas S. and Kazimier, Leonard J. Advanced COBOL. New York: McGraw-Hill, 1982.

ORGANIZATION

This document has four parts:

- Overview: Introduces Prime's COBOL 74, including supporting utilities, systems, and software (Chapter 1), and differences from the ANSI standard.

- Prime System Information: Provides information on the use of the COBOL 74 compiler (Chapter 2), and describes the process of loading and executing COBOL 74 programs (Chapter 3).

- Language Reference: Provides COBOL 74 language specifications, patterned after the ANSI standard. The three subdivisions are:

    - Elements of COBOL (Chapter 4)

    - Nucleus, Library, and Sequential I-O as one module (Chapters 5-8)

    - Other Functional Processing modules, including tape processing (Chapters 9-14)

- Appendixes and Index: The appendixes present repeatedly used data and special interfaces to other products, plus a brief conversion guide and glossary.

SUGGESTED REFERENCES

The Prime User's Guide (DOC4130) describes all supporting PRIMOS utilities for programming in Prime COBOL or any other Prime language. The Prime User's Guide is essential to the COBOL programmer.

An online system for bug reporting is available from your System Analyst. This is the POLER system.

ACKNOWLEDGMENTS

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

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

The authors and copyright holders of the copyrighted material used herein:

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

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

PRIME DOCUMENTATION CONVENTIONS

The following conventions are used in command formats, statement formats, and in examples throughout this document. Command and statement formats show the syntax of commands, program language statements, and callable routines. Examples illustrate the uses of these commands, statements, and routines in typical applications. Terminal input may be entered in either uppercase or lowercase.

Convention	Explanation	Example
UPPERCASE	In command formats, words in uppercase indicate the actual names of commands, statements, and keywords. They can be entered in either uppercase or lowercase.	SLIST
lowercase	In command formats, words in lowercase indicate items for which the user must substitute a suitable value.	CBL program-name
Underlining in examples	In examples, user input is underlined but system prompts and output are not.	OK, SEG -LOAD
Brackets [ ]	Brackets enclose a list of one or more optional items. Choose none, one, or more of these items (0 to n).	SPOOL [-LIST / -CANCEL]
Braces { }	Braces enclose a vertical list of items. Choose one and only one of these items.	CLOSE {filename / ALL}
Ellipsis ...	An ellipsis indicates that the preceding item may be repeated.	item-x[,item-y]...

Parentheses ( )	In command or statement formats, parentheses must be entered exactly as shown.	data-name (index)
Hyphen -	Wherever a hyphen appears in a command line option, it is a required part of that option.	CBL name -LIST
(CR)	Unless used in a COBOL PICTURE clause, the (CR) symbol indicates a single carriage return that is generated by pressing the RETURN key on most terminals.	
Angle brackets <>	Angle brackets must be used as shown to separate the elements of a pathname.	<FOREST>BEECH>LEAF4

ADDITIONAL DOCUMENTATION CONVENTIONS

Convention	Explanation	Example
Shading	Shading around a section of text indicates a Prime extension to or restriction on the ANSI standard.	

File-name Conventions

Convention	Explanation	Example
file-name.language or file-name	Source file	MYPROG.CBL
file-name.BIN or B_file-name	Binary (object) file	MYPROG.BIN
file-name.LIST or L_file-name	Listing file	MYPROG.LIST
file-name.SEG	Saved executable runfile (V-mode)	MYPROG.SEG

File-names may be comprised of 1 to 32 characters inclusive, the first character of which must be nonnumeric. Names should not begin with a hyphen (-) or underscore (_). File-names may be composed only of the following characters: A-Z, 0-9, _ # $ & - * . and /.

See Chapter 2 for an explanation of how the various names for source, object, listing, and runtime files relate to each other.

Note

On some devices, the underscore (_) may print as back arrow (←).

1

Overview of Prime's COBOL 74

LANGUAGE STANDARDS

Prime COBOL 74 is based upon American National Standards Institute (ANSI) COBOL, as defined in the document American National Standard Programming Language COBOL X3.23-1974, published by the American National Standards Institute, New York, 1974. Each module of the ANSI COBOL standard has two levels, as defined in X3.23-1974. Level 2 contains the full set of features; level 1 contains a subset of level 2. Appendix I shows Prime support of the ANSI COBOL 74 standard for level 2.

Prime also provides syntax checking for levels 1-4 of FIPS (Federal Information Processing) COBOL. FIPS standards are presented in Appendix C.

PRIME EXTENSIONS TO THE ANSI STANDARD

Appendix I provides a complete list of Prime extensions. In this manual, Prime extensions are indicated by shading. Some of the most important are:

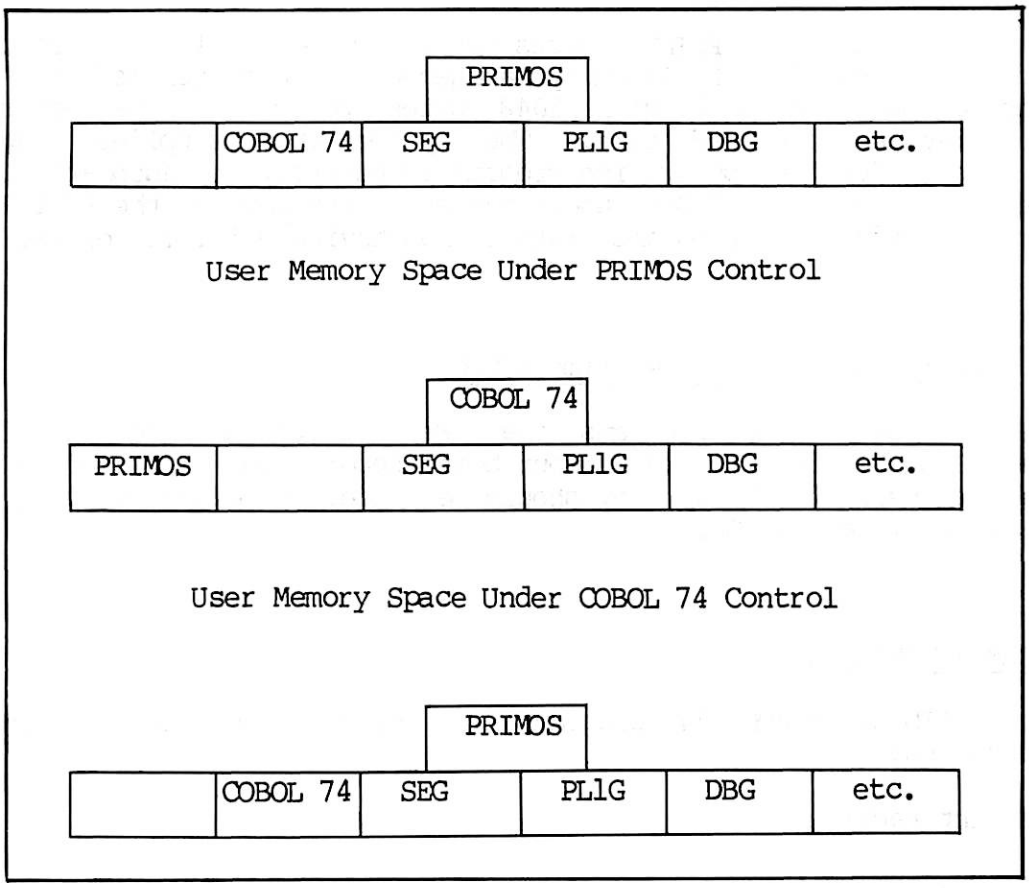
- Either the apostrophe or the single quote may be used as a delimiter for nonnumeric literals.
- Lowercase and uppercase letters can be made interchangeable except in nonnumeric literals.

- Data types COMPUTATIONAL-1, COMPUTATIONAL-2, and COMPUTATIONAL-3 (single-precision floating point, double-precision floating-point, and packed decimal) are supported.
- Paragraphs within the ENVIRONMENT division may be omitted or may be entered in any order.
- EBCDIC is supported as an alphabet-name.
- The file attribute COMPRESSED/UNCOMPRESSED is supported.
- Eight levels of subscripting are supported.
- Subscripts may themselves be subscripted, and may be arithmetic expressions.
- Arithmetic expressions may be used in place of data-names.
- The CORRESPONDING option may be used with IF and with all arithmetic verbs.
- The WITH NO ADVANCING option may be used with DISPLAY.
- THEN and OTHERWISE may be used in conjunction with IF.
- Literals may be used as operands of most clauses after INSPECT.
- The EJECT, EXHIBIT, GOBACK, NOTE, READY TRACE, REMARKS, and RESET TRACE statements have been added.
- CALL may pass arguments of a level-number other than 01 or 77 (except 66 or 88).
- The compiler may be used with options such as cross reference, data map, FIPS checking, and interface to the Prime Source Level Debugger.
- ACCEPT and DISPLAY may have operands as long as 256 characters.

COBOL 74 UNDER PRIMOS

Implementation

Prime's COBOL 74 operates under PRIMOS, the Prime operating system. Figure 1-1 is a representation of how the user's memory space appears. The first part shows how that space appears after login, when control is at PRIMOS command level. The next part shows user space after the command CBL filename has been issued and control passes to the COBOL 74 compiler. (Commands such as DBG, SEG, or PL1G cause control to pass to the Source Level Debugger, the load utility, or the PL/I Subset G compiler, respectively.) The third part represents memory after the COBOL 74 compiler has finished and has transferred control back to



Representation of the User's Memory Space as COBOL 74 is Invoked
Figure 1-1

PRIMOS. PRIMOS control is marked by the screen display OK, which may be changed by the user or by the System Administrator.

COBOL 74 runs on all Prime models that support Virtual Addressing mode (V-mode). COBOL runfiles operate in V-mode, which is explained in the Assembly Language Programmer's Guide (FDR3059-101). Prime's processors execute an extended set of instructions directly, including decimal arithmetic and character edits. They maximize execution time efficiency better than processors that only substitute an equivalent software routine. The Prime instruction set hardware is presented in the System Architecture Reference Guide (PDR3060-182) and the Assembly Language Programmer's Guide.

Operating Environment

Only one version of PRIMOS exists for all Prime models. It features paged and segmented virtual memory management. The system is based on demand paging from disk with 2048 bytes per page. A page-sharing feature reduces overhead time. The system thus supplies paging requirements for the application program immediately and automatically. For example, several COBOL users may share one copy of the EDITOR to enter or modify their programs, rather than having multiple copies.

Compatibility of Old and New Prime COBOL

The Prime system allows programs that were compiled under previous Prime COBOL compilers to run without being recompiled. If programs are recompiled using COBOL 74, they should be loaded and executed using the Rev. 18 or higher PRIMOS.

PROGRAM ENVIRONMENTS

Under PRIMOS, COBOL 74 programs may execute in one of three environments:

- Interactive
- Phantom user
- Batch job

Interactive

All phases of COBOL compilation can be handled through interactive terminals. Therefore, source programs can be entered and modified directly at a terminal. A COBOL programmer can create, edit, compile, list, debug, execute, and save a program in a single interactive session.

Program execution is initiated directly by the user. Programs run in real time and are associated with a terminal. Program output, as well as error messages, may be displayed at the terminal. Major uses are:

- Program development
- Programs requiring short execution time
- Data entry programs such as order entry or payroll
- Interactive programs such as the EDITOR

All examples of compiling, loading, and execution in this book are given for an interactive environment.

Phantom User

The phantom environment allows programs to be executed while not associated with a terminal. This frees the terminal for other uses. Phantom users are programs that accept input from a command file instead of a terminal; output directed to a terminal is either ignored or directed to a file.

Major uses of phantoms are:

- Programs requiring long execution time (such as sorts)
- Certain system utilities (such as line printer spooler)
- Any program when the terminal should be freed for another use

More detail on command files and phantom users is in the Prime User's Guide.

Batch Job

Since the number of phantom users on a system is limited, phantoms are not always available. The batch environment allows users to submit noninteractive command files as batch jobs at any time. The Batch Monitor (itself a phantom) queues these jobs and runs them, one to six at a time, as phantoms become free.

More detail on command files and batch processing is in the Prime User's Guide.

SYSTEM RESOURCES SUPPORTING COBOL 74

Prime COBOL shares with all Prime programming languages a broad range of system and file management resources. Such resources as system libraries, the text editor, the Debugger, or the SEG utility expand the scope and efficiency of Prime's interactive environment. Compatible file management systems provide standardized file management, with files created and maintained separately from the applications program. Programs compiled in any of several languages may be called by COBOL programs.

Libraries

The COBOL programmer may find system library functions and subroutines of use in some applications. A complete treatment of all library and system subroutines is in the Subroutines Reference Guide (DOC3621). A list of modules in the COBOL 74 library (CBLLIB or NCBLLIB) is in Appendix F.

Editors

Prime's EDITOR is a line-oriented text editor enabling the programmer to enter and modify source code and text files. Information for these purposes is in the Prime User's Guide. A complete description of the EDITOR is in the New User's Guide to EDITOR and RUNOFF (FDR3104-101).

EMACS is Prime's screen-oriented editor, separately priced. Information on this product is in the EMACS Primer (IDR6107-183), the EMACS Reference Guide (IDR5026), and the EMACS Extension Writing Guide (IDR5025-183).

Debugger

The Source Level Debugger is a symbolic debugger that allows breakpoints, single-stepping, tracing, change of program flow, and modification of data. The COBOL 74 compiler option that interfaces to the Debugger is presented in Chapter 2 and Appendix D. The Debugger is described in the Source Level Debugger Guide (DOC4033).

Load and Execute Utility

SEG is the loading and execution utility for COBOL and most other languages. It combines separately compiled program modules, subroutines, and libraries into an executable program. All memory management, symbol tables, and linkages are handled by SEG's loader. Various types of load maps may be obtained. The SEG utility has many functions. The minimum functions necessary for a COBOL user are described in Chapter 3 of this guide. Advanced features are presented in the SEG and LOAD Reference Guide (DOC3524-192).

Multiple Index Data Access System (MIDAS or MIDASPLUS)

MIDASPLUS (or the old MIDAS) is a system of utilities and subroutines for creating and maintaining files for use with the Indexed Sequential and Relative I-O modules of COBOL 74. MIDASPLUS provides the COBOL programmer with a transparent multilevel file structure. All housekeeping functions on the index and data subfiles are performed by

MIDASPLUS subroutines called automatically from the COBOL library routines. MIDASPLUS files created by programs written in one language may be accessed and manipulated by programs written in other languages, assuring compatibility.

The MIDASPLUS Access Manager is reentrant. All active programs on Prime models 350 and above share a single copy of the manager, minimizing redundancy. MIDASPLUS features of interest to the COBOL 74 programmer include the following.

- There can be up to 17 alternate record keys for a MIDAS or MIDASPLUS file.
- Duplicate keys let MIDAS or MIDASPLUS retrieve multiple records for a single key value.
- Keys can be constructed from concatenated information.
- A single program can make both sequential and random accesses to a single file.

Basic MIDASPLUS template construction is presented in Appendix E. The complete documentation is the MIDAS User's Guide (IDR4558), together with the update on MIDASPLUS (PTU2600-098).

Caution
Do not use COBOL with an outdated revision of MIDAS or MIDASPLUS.

Forms Management System (FORMS)

FORMS is a system for creation, maintenance, and use of screen forms for interactive file maintenance. These screen forms are an extremely useful tool for the applications programmer writing data entry programs where data fields are to be displayed in one or more formats. FORMS keeps application programs, the forms, and the devices they use separated until runtime. Thus, changes can be effected in one area without necessarily affecting the other two.

FORMS is compatible with DBMS and MIDAS or MIDASPLUS. It helps make accurate data available at widely dispersed locations for inquiry or update by all kinds of business transactions. Details are in the FORMS Programmer's Guide (PDR3040).

Language Interfaces

Since all Prime high-level languages are alike at the object-code level, object modules produced by the COBOL compiler can call and be called by modules produced by the F77, FTN, Pascal, PMA, or PL1G compilers, provided that the following restrictions are observed.

- All I-O routines must be written in the same language.
- There must be no conflict of data types for variables being passed as arguments.

For more information on language interface, see Rev. 19 or later of the Subroutines Reference Guide and Table 9-1 of this manual.

2
Compiling the Program

INTRODUCTION

To get a COBOL program running, you must first compile it successfully, which means with no errors of severity level 3 or 4. This process is explained in Chapter 2. Then you must load and execute it as explained in Chapter 3.

USING THE COMPILER

The COBOL 74 compiler is invoked by the CBL command to PRIMOS in this command line format:

```
CBL pathname [-option-1 [-option-2 ... -option-n]]
```

pathname — The pathname of the COBOL source file. Pathnames are explained in the Prime User's Guide.

options — The options controlling compiler functions such as listings and debug interface. The options are explained in COMPILER OPTIONS (COMMAND LINE OPTIONS) below. All option names must be preceded by a hyphen or dash (-).

An example is:

```
CBL MYPROG -L HOME>MYPROG.LIST
```

COMPILER OPTIONS (COMMAND LINE OPTIONS)

The compiler functions enabled by the various options of the CBL compiling command fall into six groups:

- Specifying the source file
- Specifying the existence and contents of the source listing and other listings
- Specifying the handling of error and statistics information
- Specifying the existence and properties of the object code
- Increasing segment work space
- Specifying I-O handling
- Providing help in selecting options

In this section, options that are Prime-supplied defaults are marked with bullets (o).

Note

The System Administrator may change these defaults at a particular installation. The command CBL and other features of the programming environment may also be altered.

Some options require an argument in addition to the option specification. The argument follows the option, and is not preceded by a dash. Options may be given in any order.

Table 2-1 lists the options alphabetically with their abbreviations.

The Source File

The source file is usually designated by pathname immediately after the CBL command. Alternatively, it may be given in an option. Lowercase letters in the source may be changed automatically to uppercase before compilation.

▶ -INPUT pathname
-SOURCE pathname

These are obsolete options. Either of these can be used to specify the source file to be compiled as an alternative to naming the file immediately after the CBL command.

Table 2-1
Summary of Compiler Options and Abbreviations

(Defaults are underlined.)

Option	Abbreviation	Significance
-ALLERRORS	-ALL	Produce diagnostics for all errors detected during the compilation
-BINARY o	-B	Create object file
-DEBUG	-DEB	Generate debugger code
-DIAGSONLY	-DIAGS	Produce diagnostics only (no code generation)
-ERRORFILE	-ERR	Produce error file if any diagnostics are used
-EXPLIST	-EX	Produce expanded source listing
-FIPS1		Flag all elements that exceed FIPS level 1
-FIPS2		Flag all elements that exceed FIPS level 2
-FIPS3		Flag all elements that exceed FIPS level 3
-FIPS4		Flag all elements that exceed FIPS level 4
-FORCEBINARY	-FORCE	Create object file even after fatal diagnostics
-HELP		Display list of options
-HEXADDRESS	-HEX	Print addresses in hexadecimal notation
-INPUT	-I	Designate source file
-LNKWRK2	-LN	Make the amount of work space in the linkage segment twice the default value
-LNKWRK4		Make the amount of work space in the linkage segment four times the default value
-LIST	-L	Create source listing
-MAP		Produce a data map at the end of the listing
-MAPSORT	-MAPS	Print map with names sorted alphabetically
-MAPWIDE	-MAPW	Print map and cross reference on 108-character lines
-NOBINARY	-NOBIN	Create an empty binary file
-NOCALCINDEX	-NOCALC	Calculate address of index references only at time of reference
-NOOPTIMIZE	-NOOP	Don't optimize object code
-NOOWNERID	-NOOWN	Don't enter program-id in object-program stack

Table 2-1 (continued)
Summary of Compiler Options and Abbreviations

Option	Abbreviation	Significance
-NOSYNTAXMSG	-NOSYN	Suppress syntax-recovery messages
-NOTTYDIAGS	-NOTTY	Suppress all diagnostics to the terminal
-OFFSET	-OFF	List object address of procedure statements
-OLD		Allow only those I-O constructs that were allowed in old COBOL
-OPTIMIZE ●	-OPT	Optimize object code
-PRODUCTION	-PROD	Generate production code
-RANGE		Check subscript ranges
-RMARGIN		Extend Area B to column 160
-SIGNALERRORS	-SIG	Abort execution and signal overflow errors
-SILENT	-SIL	Suppress severity-level-1 diagnostics
-SILENT2	-SIL2	Suppress level-2 diagnostics
-SILENT3	-SIL3	Suppress level-3 diagnostics
-SLACKBYTES	-SLACK	Flag each item that is compiler-aligned
-SOURCE	-S	Designate source file
-STATISTICS	-STAT	Print compiler statistics
-TOTALS	-TOT	Display program statistics
-TRUNCDIAGS	-TRUNC	Issue diagnostics for truncated result
-XREF	-X	Generate cross reference
-XREFSORT		Generate alphanumeric-order cross reference
-64V ●		Produce 64V-mode code

The options -I and -S must not be used if the source file-name immediately follows the CBL command.

▶ -RMARGIN

Extends Area B of each source line to column 160.

Existence and Contents of the Source Listing

▶ -EXPLIST

Creates a source listing with an assembly code listing.

Each statement in the source will be followed by the PMA (Prime Macro Assembler) statements into which it was compiled. For use of the listing, a knowledge of PMA is necessary. For information on PMA, see the Assembly Language Programmer's Guide.

▶ -HEXADDRESS

In conjunction with -LISTING, creates a listing file with all addresses in hexadecimal instead of octal notation.

▶ -LISTING [argument]

Specifies creation of the source listing file. The basic source listing contains the date and time of compilation, the options in effect, the source text, and a list of errors. The argument may be:

pathname	The listing will be written to the file pathname.
YES •	If the source file-name is named program.CBL, the listing file is named program.LIST. Otherwise it is named L_program. The listing is created in the UFD from which the CBL command is invoked.
TTY	The listing will be printed at the user terminal.
SPOOL	The listing will be spooled directly to the line printer.
NO	No listing file will be created.

When no -L option is given, -L NO is presumed. When -L is given with no argument, -L YES is presumed.

▶ -MAP

Produces a listing with a data map. A sample data map with discussion is given in Appendix G. This option includes -LISTING implicitly, so the two should not be used together.

▶ -MAPSORT

Produces a listing with a data map that is sorted alphabetically.

▶ -MAPWIDE

Produces a listing with a data map. The map information is printed in 108-character lines instead of 80-character lines. This option includes -LISTING implicitly, so the two should not be used together.

▶ -OFFSET

Produces a listing with the object address (octal offset from the Procedure Base register) of each PROCEDURE division statement appended in the form:

line number: halfword offset

This option includes -LISTING implicitly, so the two should not be used together.

▶ -XREF

Creates a listing with a map and cross reference. The cross reference lists, for every variable, the line number on which the variable was referenced. If the line number is preceded by an asterisk, the reference changes the variable's value. A sample cross-reference listing with discussion is given in Appendix H. This option includes -LISTING implicitly, so the two should not be used together.

▶ -XREFSORT

Creates a listing like the one generated by -XREF, but with data-names in alphanumeric order.

Error and Statistics Information

▶ -ALLERRORS

Produces diagnostics for all errors detected in the source program. If -ALLERR is not specified, compilation will abort when 100 fatal diagnostics have been issued.

▶ -ERRORFILE

If any diagnostics are issued, produces a file called program-name.CBL.ERROR, containing all diagnostics issued.

▶ -FIPS1,-FIPS2, -FIPS3, -FIPS4

Flags all occurrences of COBOL elements that exceed the specified level of FIPS (Federal Information Processing Standard). The correlation of this standard with the ANSI standard is given in Appendix C.

▶ -NOSYNTAXMSG

Suppresses the messages "SYNTAX CHECKING SUSPENDED" and "SYNTAX CHECKING RESUMED" that accompany error messages.

▶ -NOTTYDIAGS

Suppresses all diagnostics to the terminal.

▶ -SILENT

Suppresses diagnostics of severity level 1, both at the terminal and in the listing file. (Levels of messages are explained in the section COMPILER OUTPUT below.)

▶ -SILENT2

Suppresses level-2 diagnostics.

▶ -SILENT3

Suppresses level-3 diagnostics.

▶ -SLACKBYTES

Issues a severity-level-1 diagnostic for each elementary or group item that is aligned by the compiler on a 16-bit boundary. (COMP, COMP-1, and COMP-2 data items must be allocated on 16-bit boundaries. When any of these items are members of a group, they are allocated on the current available location if that location is on a 16-bit boundary. Otherwise they are shifted one byte to the right. The group item that contains the items is also shifted if required. See DATA REPRESENTATION AND ALIGNMENT in Chapter 4.

▶ -STATISTICS

Controls printout of compiler statistics.

A list of compilation statistics is printed at the terminal after each phase of compilation. Headings are:

LEX	Lexical analysis and parsing of IDENTIFICATION division
ED_PARSE	Parsing of ENVIRONMENT division
DD_PARSE	Parsing of DATA division
PD_PARSE	Parsing of PROCEDURE division
ALLOCATOR	Data allocation
GENERATOR	Optimization and object-code generation
TOTAL	Total disk time for all phases

For each phase the list contains:

DISK	Number of reads and writes during the phase, excluding those needed to obtain the source file
SECONDS	Elapsed realtime
SPACE	Internal buffer space used for symbol table, in units of 16K bytes
PAGING	Disk I-O time
CPU	CPU time in seconds, followed by the clock time when the phase was completed

At the end of the listing, storage allocation statistics are appended. All sizes are stated in 16-bit halfwords. The statistics are:

Code Size	The number of halfwords of object code generated from the PROCEDURE division of the source program. The maximum allowable is listed in Appendix J.
Static Size	The size of the user-defined WORKING-STORAGE.
Source Lines	The number of lines in the source program.
Lines per Min	Lines compiled per minute.

▶ -TOTALS

Displays the following program statistics on the terminal at the end of compilation:

- Procedure code size in 16-bit halfwords. This is the part of the program addressed by the procedure base register, explained in the Assembly Language Programmer's Guide.
- Static (WORKING-STORAGE) size in 16-bit halfwords. This is the part of the program addressed by the link base register, which is explained in the Assembly Language Programmer's Guide.
- Number of source lines.
- Compilation speed in lines compiled per minute.

▶ TRUNCDIAGS

Issues diagnostics whenever the result cannot fit in the receiving field.

Existence and Properties of the Object Code

▶ -BINARY [argument] •

Specifies a binary (object) file-name to override the default. The argument may be:

pathname	Object code is written to the file pathname.
YES •	If the source file-name is program.CBL, the object file is named program.BIN. Otherwise the object code is written to a file named B_program. The file is created in the UFD from which the CBL command is invoked.
NO	No binary file is created. Specified when only syntax check or source listing is desired.

When no -B option is given, or -B without an argument is given, -B YES will be presumed.

More information on naming binary object files is given below in COBOL FILES -- NAMING CONVENTIONS.

Storage Allocation and Addressing: The programmer can specify the addressing mode (64V) to be used in the object file.

▶ -64V •

This option specifies the addressing mode to be used in the object code. 64V is a segmented virtual addressing mode for 32-bit machines. It is the default and only storage option available for COBOL.

Augmented Object Code

▶ -DEBUG

Controls generation of code for the Debugger.

The object file is modified so that it will run under the Source Level Debugger. Execution time is increased. The code is not optimized.

When the -DEBUG option has been included in the command line, the resulting object file can be debugged using the commands specified in the Source Level Debugger Guide. Appendix D gives more information on COBOL 74 for the Debugger.

▶ -DIAGSONLY

Compiles for diagnostics only; does not execute the code generation phase and produces only an empty binary file.

▶ -FORCEBINARY

Creates a binary file even if fatal diagnostics were issued.

▶ -NOCALCINDEX

Performs address calculations for indexed references at the time the indexed item is referenced, instead of when the index is changed by a SET, SEARCH, or PERFORM statement.

▶ -NOOWNERID

Does not enter the program-id onto the object-program stack, thus saving a small code sequence for extremely time-critical programs. (The program-id on the stack is useful with the PRIMOS command DMSTK for debugging.)

▶ -OPTIMIZE •

Controls the optimization phase of the compiler.

The object code will be optimized. Optimized code runs more efficiently than nonoptimized code, but takes longer to compile. It performs such automatic tasks as keeping track of register contents and evaluating constant expressions.

▶ -NOBINARY

Creates only an empty binary file. This option has the same effect as -DIAGSONLY.

▶ -NOOPTIMIZE

Optimization does not occur. This saves some compilation time.

▶ -PRODUCTION

Controls code for the Debugger.

-PRODUCTION is similar to -DEBUG, except that the code generated will not permit insertion of statement breakpoints. Execution time is not affected.

▶ -RANGE

Controls error checking for out-of-bounds values of array subscripts.

Error-checking code is inserted into the object file. Should an array subscript take on a value outside the range specified in the DATA entry, the ERROR condition will be signalled. (Note that range checking decreases the efficiency of the generated code.)

▶ -SIGNALERRORS

Aborts execution and signals a condition for arithmetic overflow and conversion errors. Exponentiation errors always abort object execution.

Increase Segment Work Space

▶ -LNKWRK2
-LNKWRK4

Makes the amount of work space available in the linkage segment, needed by the code generator, either twice or four times the default value. Use this switch if you get the compile time message, "The amount of compiler work space exceeds the current limit."

Specify I-O Handling

▶ -OLD

Allows compatibility with I-O constructs that were allowed in old COBOL:

- Requires manual execution-time assignment of files unless the EXIT PROGRAM statement is used. (See Appendix K.)
- Uses COMPRESSED as the default FD option.
- Allows only one through eight characters in the literal or data-name for FILE-ID. (Normal I-O allows 1-120 character values.)
- Supplies a file-name in the series F1, F2, etc., if no VALUE OF FILE-ID clause is present for an FD. (Normal I-O uses the file-name in the associated SELECT clause.)
- Allows only one through six characters in the literal in the OWNER clause. (Normal I-O allows 1-120 characters.)

Provide Help

▶ -HELP

Displays a list of compiler options and their functions.

This option should be entered with no operands, as follows:

```
CBL -HELP
```

COBOL FILES -- NAMING CONVENTIONS

File Types

Three types of files may be involved in compilation: source file, listing file, and object file. Of these, the listing and object files are compiler-generated. Corresponding PRIMOS file units are given in Table 2-2 below. (File unit numbers, which are needed for some PRIMOS commands, are explained in Chapter 3 of the PRIMOS Commands Reference Guide.)

Table 2-2
PRIMOS File Units

File Type	PRIMOS File Unit
Source	1
Listing	2
Object	3

File-names

If a file-name is specified in the compile command line for the listing or object file, the COBOL compiler causes these files to be opened under the file-name specified. If no file-name is used for the object or listing file, two default options are possible.

Normal Default Naming: The COBOL 74 compiler first looks for a file with the suffix .CBL, then for the file-name alone. Thus, it is only necessary to use the file-name without the suffix when invoking CBL. If the source program name ends in .CBL, the default binary file-name will be the file-name plus the suffix .BIN. If requested, the default listing name will be the file-name plus the suffix .LIST and the error file will be file-name.CBL.ERROR.

Thus, for MYPROG.CBL, if the compile command line is:

```
CBL MYPROG -LIST -ERRORFILE
```

the files produced will be MYPROG.BIN, MYPROG.LIST, and MYPROG.CBL.ERROR. The object file may then be used for the default loading described in Chapter 3.

The Older Naming Convention: An older naming convention is followed if the source file-name does not end in .CBL. The default convention for a listing file is L_file-name. The default convention for an object file is B_filename. An error file named file-name.CBL.ERROR is also created, if requested.

Thus, for a source file named SAM, following the compile command CBL SAM -LIST -ERRORFILE, the listing and object files would exist in the current UFD as L_SAM and B_SAM, respectively. If errors are generated, they may be recorded in SAM.CBL.ERROR.

Default UFDs: If the source file is given as a pathname such as [<MFD>]UFD1 ...>SAM, where the file SAM does not reside in the current UFD (that in which compilation is occurring), the listing and object files will, nevertheless, be created in the current UFD unless another UFD is specified. This is true for both naming conventions.

Table 2-3 summarizes the two naming conventions.

Table 2-3
Default Naming Conventions

Source	Binary	List	Errors
file-name	B_file-name	L_file-name	file-name.CBL.ERROR
file-name.CBL	file-name.BIN	file-name.LIST	file-name.CBL.ERROR

Setting Default Names from PRIMOS: If the user desires the listing or object files to have default names other than those outlined above, the PRIMOS commands LISTING or BINARY with arguments must be invoked prior to compilation. These command are discussed in the PRIMOS Commands Reference Guide.

COMPILER OUTPUT

The compiler, as default options, sends output to the terminal and to a binary file. If the number of fatal errors exceeds 100, compilation ends unless the -ALLERRORs option was specified. Compiler file output is summarized in Table 2-4.

Output to the Terminal

If compilation is successful, the screen will display a message in the following format:

```
[CBL rev x.x ]
OK,
```

Table 2-4
Compiler File Specifications

(X means the argument must not be used with this option.)

Argument Following Option	Option		
	-INPUT or -SOURCE	-LISTING	-BINARY
pathname	Looks for file named pathname.CBL, then pathname as source file	Opens file named pathname as listing file	Opens file named pathname as binary (object) file
YES	X	Uses default file-name for listing file: PROGRM.LIST or L_PROGRM	Uses default file-name for binary file: PROGRM.BIN or B_PROGRM
NO	X	No listing file	No binary file
TTY	X*	Print listing on user terminal	X
SPOOL	X	Spool listing directly to printer queue	X
Option not invoked	Source file-name should be first option after CBL command*	Same as NO	Same as YES

* Other options are possible but not recommended for the COBOL programmer.

If errors occur during compilation, error messages are output to the terminal, and to the error file and the listing file if they have been specified. An example is:

```
CBL MYPROG
[CBL rev x.x ]
ERROR 7  SEVERITY 3  LINE 27  COLUMN 36  [FATAL, SEMANTICS]
Picture clause too long.
ER!
```

Note

If the compiler error message includes FATAL, no object file is produced.

After compilation, control returns to PRIMOS.

Compiler Error Messages

The general format of the error message is:

```
ERROR e  SEVERITY s  LINE l  COLUMN c  [  text    ]
diagnostic
```

The elements of this format are:

e	The COBOL error number	
s	The severity number:	
	1	Observation
	2	Warning
	3	Fatal (no object code produced)
	4	Abortion of compilation (no object code produced)
[text]	A description of the severity level and type:	
	SYNTAX	Usually fatal, caused by violation of syntax rules or format
	SEMANTICS	Caused by violation of syntax rules or general rules
diagnostic	The COBOL compiler error message	

3

Loading and Executing Programs

After a program has been compiled as discussed in Chapter 2, it must be loaded into an executable file before being run or executed. The PRIMOS SEG utility loads and executes all COBOL 74 programs. The loading steps create a runfile, or executable file consisting of one or more object programs plus any necessary subroutines and libraries. This runfile, or run unit, can then be executed at will. This chapter describes normal loading and execution, and specifies some techniques required for switch settings. Loading is described in more detail in the Prime User's Guide. For extended loading features and a complete description of all SEG commands, including those for system-level programming, refer to the SEG and LOAD Reference Guide.

LOADING PROGRAMS

Default Loading

On Rev. 18 and higher, the SEG utility can create a default runfile named program.SEG and load default object file-names. Use the -LOAD parameter after SEG, providing the object filename ends in .BIN.

1. Give the command SEG -LOAD. The response will be a dollar sign ($), indicating that the load subprocessor is ready.

2. Use the LOAD command with either the binary filename or the source filename. In the latter case, SEG will look for a binary file of the same name, followed by .BIN.

3. Use the LOAD command to load the object files of any separately compiled subroutines (preferably in order of frequency of use).

4. Use the LIBRARY command to load subroutines called from libraries in the following order:

   - The COBOL 74 library (CBLLIB or NCBLLIB)
   - The sort-merge library, if sort-merge files are loaded (VSRTLI, or NVSRTLI if a nonshared library is needed)
   - Other Prime libraries, if required (filename)
   - The PRIMOS system subroutine library -- required (LI with no filename)

   See Figure 3-1 for a diagram of this process.

   At this point, you should receive a LOAD COMPLETE message. If the message is absent, check whether any required libraries, programs to be called, or subroutines are missing. If necessary, enter MAP 3 (described in the Prime User's Guide and the LOAD and SEG Reference Guide) to identify the unsatisfied references and load them. If the unsatisfied references are caused by missing subroutine names, enter QUIT and restart from Step 1. If some other SEG error message appears, refer to the LOAD and SEG Reference Guide for the probable cause and correction.

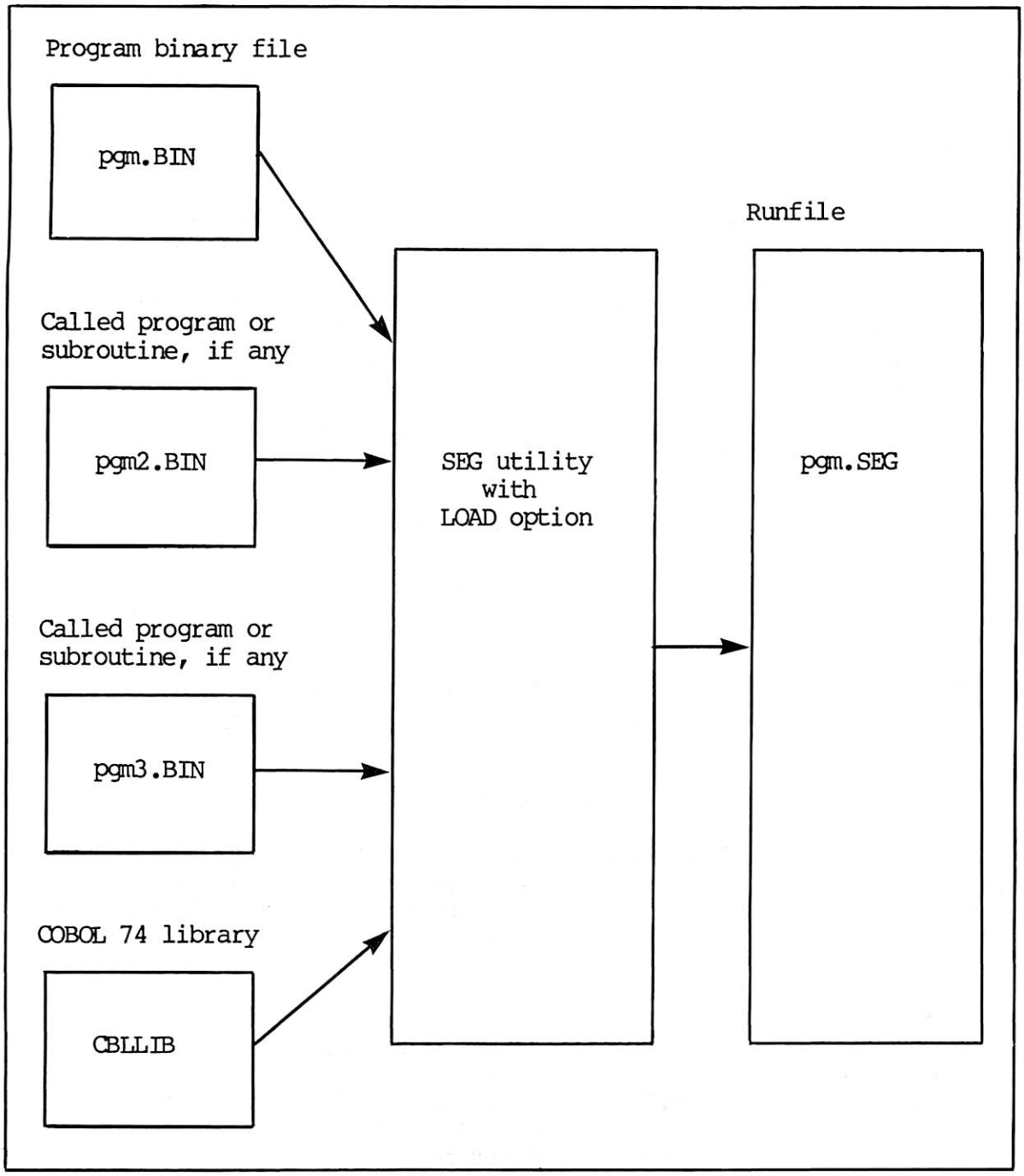
5. Enter QUIT to save the runfile and exit from the utility.

   SEG will give the runfile the default name filename.SEG, where filename is the name of the first object file loaded.

As an example, suppose you have a main program called MYPROG.CBL, compiled to produce an object file called MYPROG.BIN. The runfile can be created as follows:

```
OK, SEG -LOAD
[SEG rev x.x]
$ LO MYPROG
$ LI CBLLIB
$ LI
LOAD COMPLETE
$ Q
OK,
```

The command LO MYPROG loads MYPROG.BIN. The resulting runfile is automatically named MYPROG.SEG.



Loading (Creating a Runfile or Run Unit)
Figure 3-1

The Older Loading Procedure

On Rev. 19 and lower, loads can be accomplished by the following procedure:

1. Invoke the SEG loader with the SEG command without options. A pound sign (#) will be the response and prompt symbol.

2. Enter the SEG-level LOAD command to start the load subprocessor and to set up the runfile with a name selected by you (LO runfilename). A dollar sign will appear as the next prompt symbol.

3. Use the LOAD command to load the object files in the following order:

   - The object file of the main program (B_filename)
   - The object files of any separately compiled programs or subroutines to be called (preferably in order of frequency of use)

4. Use the LIBRARY command to load subroutines called from libraries in the same order as in Step 4 for Default Loading above.

5. Enter QUIT to save the runfile and exit from the utility.

As an example of loading, assume that the user has compiled a main program, MAIN, and a subroutine in a separate source file, SUBR. Both have been compiled using the default object file-names B_MAIN and B_SUBR. They could be loaded as follows:

```
OK, SEG                 Brings SEG into memory
[SEG rev 18.X]
# LO MAIN.SEG           Invokes the loader and establishes a runfile
$ LO B_MAIN             Loads the main program
$ LO B_SUBR             Loads any separately compiled subroutine
$ LI CBLLIB             Loads the COBOL 74 library
$ LI                    Loads the subroutine library
LOAD COMPLETE           Loader indicates all references are satisfied
$ Q                     Returns to PRIMOS level
OK,
```

Note

Any name may be supplied for the runfile. A name ending with .SEG is suggested to identify runfiles, and to allow the default execution method described below.

Load Error Messages

If the message WARNING -- LOAD NOT COMPLETE is displayed, the cause may be:

- Not loading necessary libraries such as VSRTLI or CBLLIB
- Not loading a program named in a CALL statement

To list all unresolved references, go through the loading routine and, after the final LI, enter MAP 3. The MAP command is discussed in the LOAD and SEG Reference Guide.

EXECUTING LOADED PROGRAMS -- RUNTIME

Any of the following methods will start program execution, including any necessary requests for switch settings. (See the following sections.)

Executing Default Runfiles

If the runfile name ends in .SEG, you can execute the runfile by using only the source program name, because, given pathname, SEG looks first for pathname.SEG, then for pathname. For the default runfile MYPROG.SEG in the previous example, execution is accomplished with:

```
SEG MYPROG
```

Execution of Other Runfiles

For runfiles whose names do not end in .SEG, execution is performed at the PRIMOS level using the SEG command:

```
SEG runfilename
```

where runfilename is the pathname of a runfile created as described in The Older Loading Procedure above, but whose name does not end with .SEG.

Immediate Execution

A shortcut to saving and executing a loaded program is available. In the loading process described in the previous sections, immediately after receiving the LOAD COMPLETE message, enter EXECUTE. This command will then save the loaded program and start executing the program. The runfile is automatically saved. EXECUTE may be used only within the SEG subprocessor environment (that is, when the prompt $ is displayed).

Upon completion of program execution, control returns to PRIMOS command level.

Printer Output File-name Conventions

Print files follow another default naming convention. Files assigned to the printer are assigned by the COBOL runtime package to names consisting of the first four characters of the program-id plus a two-digit sequence number. The sequence number is 01 the first time the program is run. It is incremented by one each time the program is executed, if previous print files still exist. Thus when the program containing the program-id DISBURSE is run the first time, the print file is named DISB01 by default. The second run creates DISB02 if DISB01 still exists.

SWITCH SETTINGS AT RUNTIME

If any of the switch-names CBLSW0 through CBLSW7 (defined in Chapter 6) have been used in the COBOL source, execution will include a request for switch settings, in this format:

```
SPECIFY ON SWITCHES:
```

The user should enter the numbers of all COBOL switches, from 0 through 7, that are to be on during this execution. The numbers must be separated by either spaces or commas. If an entry is wrong, an error message is displayed and the request is repeated.

As an example, for the sample program given in the SPECIAL-NAMES section of Chapter 6, if no tape processing or printout is desired, use the following dialog at runtime:

```
SPECIFY ON SWITCHES:
1
```

RUNTIME ERROR MESSAGES

COBOL runtime error messages are self-explanatory. System runtime error messages are in the Prime User's Guide.

Common System Runtime Messages

The following system error conditions are typical of those that can occur during execution of a COBOL program.

ACCESS_VIOLATIONS$ — The run unit or runfile attempted to violate the CPU access rules. This condition aborts the run unit and can be caused by a variety of factors. One common cause is reference to a table with an out-of-range subscript. Use the -RANGE option to locate the statement that caused the subscript to go out of range.

ARITH$ — An arithmetic exception involved data overflow of fixed or floating point operands. This condition can occur only if the -SIGNALERRORS option was specified, or for an exponential operation. The run unit aborts. If -SIGNALERRORS was not specified, execution of the run unit continues with truncation of the value that would have caused the exception.

ERROR$ — A conversion involving illegal characters was attempted. Implicit conversions can occur in COBOL programs when fields of different types are moved. The run unit aborts. In general, this condition will not occur unless the -SIGNALERRORS compile-time option was specified. Recompiling the program with the READY TRACE statement, or recompiling with the -DEBUG option and then executing the run unit under Debugger control, will help locate the offending conversion.

LINKAGE_FAULT$ — An unsnapped link (unresolved call) was encountered but the reference could not be found in the system entry point table. The run unit aborts. Either not enough libraries were loaded or a program referenced in a COBOL CALL statement was not loaded when this run unit was linked with the SEG utility. If a map was created when the run unit was linked, check it for unresolved entries. Otherwise invoke SEG again and, after LI, enter MAP 3.

OUT_OF_BOUNDS$ — See ACCESS_VIOLATION$.

POINTER_FAULT$ — A reference has been made to an indirect pointer (IP) but the pointer does not appear to be valid. The run unit aborts. The most likely cause is a link base that has been destroyed by a MOVE statement with out-of-range subscripts. Recompile with the -RANGE option.

4

Elements of Prime COBOL 74

DIVISIONS OF A COBOL PROGRAM: A SUMMARY

Every COBOL program consists of four divisions:

- IDENTIFICATION division
- ENVIRONMENT division
- DATA division
- PROCEDURE division

IDENTIFICATION Division

The IDENTIFICATION division (ID division) assigns a name to the program and allows the programmer to enter other information, such as the programmer's name, the date the program was written, and remarks.

ENVIRONMENT Division

The ENVIRONMENT division specifies those aspects of a program that depend upon the physical characteristics of a specific computer, its peripheral devices, and file system. Two sections make up the ENVIRONMENT division: the CONFIGURATION section and the INPUT-OUTPUT section.

The CONFIGURATION section describes the computer on which the source program is compiled and the computer on which the compiled program is to be run. It also relates implementor-names used by the compiler to names introduced by the programmer in the source program.

The INPUT-OUTPUT section describes each file, and associates the file with a peripheral device or a storage medium.

DATA Division

The DATA division provides the compiler with a description of every data item used within the program. There are three sections of the DATA division: the FILE section, the WORKING-STORAGE section, and the LINKAGE section.

The FILE section describes the structure of data files. Each file is defined by a file-description entry and one or more record-description entries.

The WORKING-STORAGE section describes records and noncontiguous data items that are not part of external files, but are developed and processed internally.

The LINKAGE section is meaningful only in a called program. This section describes data items that may be used by both the called and calling programs.

PROCEDURE Division

The PROCEDURE division contains instructions (COBOL statements) to solve a data processing problem. This division contains two types of sections: declarative sections and procedural sections. The maximum PROCEDURE division size is listed in Appendix J.

Declarative sections contain instructions that are not performed in the regular sequence of coding. Such procedures are usually executed when an error condition is detected during a file operation.

Procedural sections contain zero or more paragraphs each. Each paragraph consists of a paragraph-name followed by zero or more COBOL sentences. Sentences, in turn, are comprised of one or more COBOL statements. Sections and paragraphs are optional.

Execution of the instructions in the PROCEDURE division begins with the first statement in the division, excluding declaratives. Statements are executed in the order in which they are written in the source program, until a PERFORM, GO TO, or other transfer of control is encountered.

Program Outline and Example

The following outline defines the program format and order (conventions of notation are explained on page 4-5):

```
{ IDENTIFICATION DIVISION. }
{ ID DIVISION.             }

  PROGRAM-ID. program-name.
  [AUTHOR. [comment-entry] ... ]
  [INSTALLATION. [comment-entry] ... ]
  [DATE-WRITTEN. [comment-entry] ... ]
  [DATE-COMPILED. [comment-entry] ... ]
  [SECURITY. [comment-entry] ... ]
  [REMARKS. [comment-entry] ... ]
[ ENVIRONMENT DIVISION.
  [ CONFIGURATION SECTION.
    [SOURCE-COMPUTER. source-computer-entry.]
    [OBJECT-COMPUTER. object-computer-entry.]
    [SPECIAL-NAMES. [special-names-entry.] ... ]
  [ INPUT-OUTPUT SECTION.
    [FILE-CONTROL. {file-control-entry}] ...
    [I-O-CONTROL. [I-O-control-entry]]
  ]
]
  DATA DIVISION.
[ FILE SECTION.
  [ file-description-entry.
    [record-description-entry] ... ] ...
  [ sort-file-description-entry.
    {record-description-entry} ... ] ...
]
[ WORKING-STORAGE SECTION.
  [ level-77-description-entry ] ...
  [ record-description-entry   ]
]
[ LINKAGE SECTION.
  [ level-77-description-entry ] ...
  [ record-description-entry   ]
]
  PROCEDURE DIVISION [USING data-name-1 [,data-name-2] ... ].
[ DECLARATIVES.
  {section-name SECTION [segment-number]. USE-sentence.
  [paragraph-name. [sentence] ... ] ... } ...
  END DECLARATIVES.
]
[ [section-name SECTION [segment-number].]
  [ paragraph-name.] [sentence] ... ] ...
]
```

The following listing file for program SAMPLE illustrates COBOL program format and order. SAMPLE reads a file and prints its contents.

```
Source File:  <OPERSY>ANNE.K>PEGS-SAMPLE.CBL
Compiled on:  THU, SEP 23 1982 at 13:27  by: CBL rev x  06/09/82.09:07:
44.Wed
```

Options are: LISTING OPTIMIZE U(PPER)CASE

```
1            ID DIVISION.
2            PROGRAM-ID. PEGS-OWN.
3            INSTALLATION. PRIME 50 SERIES.
4            DATE-WRITTEN. FEB. 25,1982.
5            DATE-COMPILED. 820923.13:28:00.
6            SECURITY. NONE
7            REMARKS. THIS PROGRAM READS A FILE AND PRINTS ITS
8                CONTENTS SEQUENTIALLY.
9           ******************************************************
10           ENVIRONMENT DIVISION.
11           CONFIGURATION SECTION.
12           SOURCE-COMPUTER. PRIME-750.
13           OBJECT-COMPUTER. PRIME-750.
14           INPUT-OUTPUT SECTION.
15           FILE-CONTROL.
16               SELECT PRINT-FILE ASSIGN TO PRINTER.
17               SELECT INPUT-FILE ASSIGN TO PFMS.
18          ******************************************************
19           DATA DIVISION.
20           FILE SECTION.
21           FD  PRINT-FILE, LABEL RECORDS ARE OMITTED,
22               DATA RECORD IS PRINT-LINE,
23               VALUE OF FILE-ID IS FILE-NAME-PT.
24           01  PRINT-LINE.
25               05  FILLER              PIC X.
26               05  OUT-LINE            PIC X(72).
27           FD  INPUT-FILE, LABEL RECORDS ARE STANDARD,
28               VALUE OF FILE-ID IS 'IN-DATA',
29               DATA RECORD IS INPUT-IMAGE.
30           01  INPUT-IMAGE             PIC X(72).
31          *
32           WORKING-STORAGE SECTION.
33           01  HEADER.
34               05  FILLER              PIC X    VALUE SPACES.
35               05  H1                  PIC X(71)
36                    VALUE 'NAME                              STREET
37          -    '      CITY          '.
38           77  FILE-NAME-PT            PIC X(8) VALUE 'PEGS.RPT'.
39           77  NO-MORE-INPUTS          PIC X    VALUE 'N'.
40          ******************************************************
41           PROCEDURE DIVISION.
42          *
43           DECLARATIVES.
44           INPUT-ERROR SECTION. USE AFTER ERROR PROCEDURE ON
45                 INPUT-FILE.
46           ONLY-PARAGRAPH. DISPLAY 'ERROR ON WRITE'.
47                 CLOSE INPUT-FILE, PRINT-FILE.
48                 STOP RUN.
49           END DECLARATIVES.
50          *
51           BEGINNING SECTION.
52             100-CREATE-FILE.
```

```
53              OPEN INPUT INPUT-FILE.
54              OPEN OUTPUT PRINT-FILE.
55              PERFORM 300-NEW-PAGE.
56              PERFORM 150-READ-PRINT.
57              PERFORM LAST-SECTION.
58              STOP RUN.
59            150-READ-PRINT.
60              READ INPUT-FILE AT END MOVE 'Y' TO NO-MORE-INPUTS.
61              PERFORM 155-PROCESS UNTIL NO-MORE-INPUTS = 'Y'.
62          155-PROCESS.
63              MOVE INPUT-IMAGE TO OUT-LINE.
64              WRITE PRINT-LINE.
65              READ INPUT-FILE AT END MOVE 'Y' TO NO-MORE-INPUTS.
66          LAST-SECTION SECTION.
67            200-CLOSE-ALL.
68              CLOSE INPUT-FILE, PRINT-FILE.
69              DISPLAY 'END OF FILE'.
70          THIRD-LEVEL SECTION.
71            300-NEW-PAGE.
72              MOVE SPACES TO PRINT-LINE.
73              WRITE PRINT-LINE FROM HEADER AFTER ADVANCING PAGE.
74              MOVE SPACES TO PRINT-LINE.
75              WRITE PRINT-LINE AFTER ADVANCING 1.
```

FORMAT NOTATION

Throughout this document, formats are prescribed for various clauses or statements. They are presented in ANSI COBOL notation, except for the use of brackets and the Prime extensions discussed below.

ANSI Notation

Words: All underlined uppercase words are called keywords and are required when the clauses containing them are used. Uppercase words that are not underlined are optional. Uppercase words, whether underlined or not, must be spelled correctly.

Lowercase words are generic terms used to represent COBOL words, literals, PICTURE character-strings, comment-entries, or a complete syntactical entry that must be supplied by the user. Where generic terms are repeated, a number appendage to the term identifies that term.

Level-numbers: When specific level-numbers appear in data-description-entry formats, those specific level-numbers are required. In this document, the forms 01, 02, and so on are used to indicate level-numbers 1 through 9.

Brackets and Braces: When a portion of a general format is enclosed in brackets, [], that portion may be included or omitted at the user's choice. Braces, {}, enclosing a portion of a general format mean that a selection of one of the options contained within the braces must be made. In both cases, a choice is indicated by vertically stacking the possibilities. However, if the items within brackets themselves are enclosed in brackets, then the header of that section or division is required if any other items are used. If a line within brackets is indented, it is part of the preceding line. When brackets or braces enclose a portion of a format, but only one possibility is shown, the function of the brackets or braces is to delimit that portion of the format to which a following ellipsis applies. If an option within braces contains only reserved words that are not keywords, then the option is a default option (selected unless one of the other options is explicitly indicated).

The Ellipsis: In the general formats, the ellipsis represents the position at which repetition may occur at the user's option. The portion of the format that may be repeated is determined as follows: scanning right to left, determine the] or } immediately to the left of the ellipsis. Continue scanning right to left and determine the matching [or {; the ellipsis applies to the words between this matching pair of delimiters.

Format Punctuation: The punctuation characters comma and semicolon are shown in some formats. They are optional and may be included or omitted by the user. In the source program these two punctuation characters are interchangeable. Neither one may appear immediately preceding the first clause of an entry or paragraph.

If desired, a semicolon or comma may be used between statements in the PROCEDURE division.

Paragraphs within the IDENTIFICATION and PROCEDURE divisions, and the entries within the ENVIRONMENT and DATA divisions, must be terminated by the period.

Special Characters: The characters +, -, >, <, =, when appearing in formats, although not underlined, are required.

Examples: In the formats on page 4-3, PROGRAM-ID is a keyword that must be used, and it must be followed by either a data-name or a literal to be supplied by the programmer. The sort file-description-entry is optional. If used, it must be followed by one or more record descriptions, which are described in Chapter 7, THE DATA DIVISION. The entire ENVIRONMENT division is optional; however, if CONFIGURATION SECTION is included, ENVIRONMENT DIVISION must be included.

Prime Extensions to ANSI Notation

The following Prime terms are added to ANSI notation.

The term data-name means a user-defined name for a variable that may be subscripted or qualified.

Clause, Statement, Entry: Certain entries in the formats consist of one or more capitalized words followed by the word "Clause," "Statement," or "Entry." These designate clauses or statements described in other formats in appropriate sections of the text.

Hyphens: For easier reference in the text, some lowercase words are followed by a hyphen and a digit or letter. This modification does not change the syntactical definition of the word.

Multiple Formats: For a given COBOL verb, separate formats are mutually exclusive options.

CODING RULES

Program layouts must follow these rules, which are illustrated in Figure 4-1.

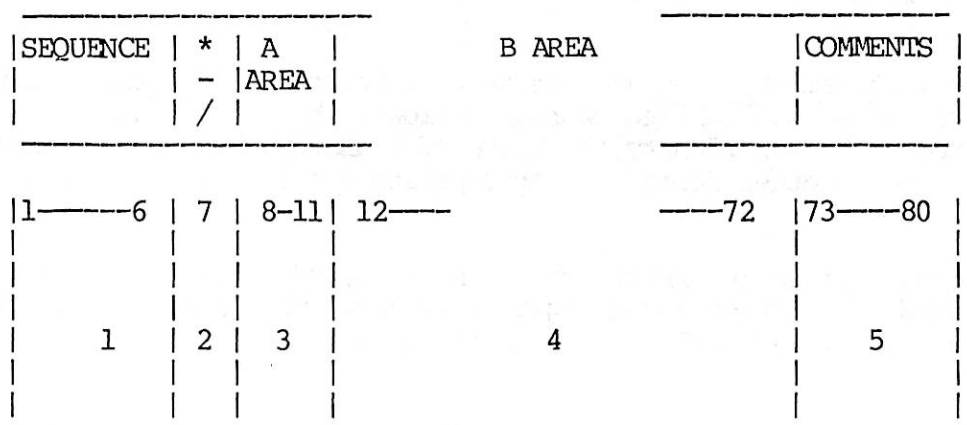
1. Each line of code may have a six-digit sequence number in positions 1-6, such that the source statements are in ascending order. Blanks are also permitted in positions 1-6.

2. Position 7 is used for four special coding symbols. An asterisk (*) in position 7 of the line causes that line to be treated as a comment. Any characters may follow on that line. The asterisk and the characters will be produced on the source listing but serve no other purpose. If a slash (/) appears in position 7, the current line is treated as a comment line, and the next line will be printed at the top of a new page of the compiler-generated listing. A hyphen (-) is used to continue any word or literal from one line to another. Refer to Continuation of Literals in this chapter for continuation rules. A "D" causes the line to be executed from DBG only.

3. Division, section, and paragraph headers must begin in the A Area (positions 8-11). Paragraph-names must also appear in the A Area (at the point where they are defined). All level-numbers may appear in the A or B Area.

4. All other program elements must be confined to Area B.

5. Positions 73-80 are ignored by the compiler unless the -RMARGIN option is in effect. Frequently, these positions are used to contain the program identification.



Standard COBOL Coding Areas
Figure 4-1

PUNCTUATION AND SEPARATORS

A separator is a string of one or more punctuation characters. The rules for formation of separators are:

- The punctuation character space is a separator. Anywhere a space is used as a separator, more than one space may be used.

- The punctuation characters comma, semicolon, and period, when immediately followed by a space, are separators. These separators may appear in a COBOL source program only where explicitly permitted by the general formats, by format punctuation rules, by statement and sentence structure definitions, or by format rules. The period followed by a space also serves as a statement terminator for conditionals.

- The punctuation characters right and left parenthesis are separators. Parentheses may appear only in balanced pairs of left and right parentheses delimiting subscripts, indexes, arithmetic expressions, or conditions.

- The punctuation character quotation mark is a separator. An opening quotation mark must be immediately preceded by a space or left parenthesis; a closing quotation mark must be immediately followed by a space, comma, semicolon, period, or right parenthesis. Quotation marks may appear only in balanced pairs delimiting nonnumeric literals except when the literal is continued. (See Continuation of Literals below.)

- The pseudo-text delimiter == is a separator. An opening pseudo-text delimiter must be immediately preceded by a space; a closing pseudo-text delimiter must be immediately followed by one of the separators space, comma, semicolon, or period. See COPY in Chapter 8. Pseudo-text delimiters may appear only in balanced pairs delimiting pseudo-text with COPY...REPLACING.

- The separator space is optional immediately preceding all separators except:

    - As specified by format rules.

    - When the separator is a closing quotation mark. In this case, a preceding space is considered part of the nonnumeric literal and not a separator.

    - Before the opening pseudo-text delimiter, where the preceding space is required.

- The separator space may immediately follow any separator except the opening quotation mark. In this case, a following space is considered as part of the nonnumeric literal and not as a separator.

Any punctuation character that appears as part of a PICTURE character-string or numeric literal is not considered a punctuation character, but rather an element of that PICTURE character-string or numeric literal. PICTURE character-strings are delimited only by the separators space, comma, semicolon, or period.

The rules established for the formation of separators do not apply to the characters that make up the contents of nonnumeric literals, comment-entries, or comment lines.

The standard character set used by Prime is the ANSI, ASCII, eight-bit character set. The set of characters, with octal, hexadecimal, and punched card equivalents, is presented in Appendix A.

THE PRIME COBOL 74 CHARACTER SET

The standard COBOL language character set, shown in Table 4-1, has 51 characters as follows: the numbers 0 through 9, the 26 uppercase letters of the English alphabet, the space (blank), and 14 special characters.

Table 4-1
Prime COBOL 74 Character Set

Class	Character	Meaning
Numeric	0...9	Digit
	figurative constants LOW-VALUES,ZERO,ZEROS,ZEROES	Values null and zero
Alphabetic	A...Z	Uppercase letters
	a...z	Lowercase letters
	Space	Blank
	Figurative constant SPACE(S)	Value blank
Special characters	+	Plus sign
	-	Minus sign
	_	Underscore
	*	Asterisk (star)
	=	Equal sign
	$	Currency sign
	,	Comma
	;	Semicolon
	.	Period
	"	Quotation mark
	'	Apostrophe
	(	Left parenthesis
	)	Right parenthesis
	>	Greater than
	<	Less than
	/	Slash (stroke)
	Figurative constants QUOTE(S),HIGH-VALUE(S)	Values quotation and delete

Prime Extensions

Prime extends the set by allowing the 26 lowercase letters and the underscore character to be used.

> **Caution**
>
> This extension does not apply to the contents of fields at runtime. In particular, lowercase letters are not considered as alphabetic in class tests.

A single quote (apostrophe) is accepted as an equivalent of double quote marks. Quotes preceding and following a given item must be identical. (Note that double quote marks and the question mark may be used in entering COBOL programs with the EDITOR only if other deletion characters have been established with the PRIMOS command TERM or the EDITOR command SYMBOL.) The Prime COBOL 74 character set is illustrated in Table 4-1.

Collating Sequence

Each character in the Prime character set has a unique value that establishes the collating sequence for the character set. This sequence conforms to the American Standards Code for Information Interchange (ASCII). The characters are arranged in ascending ASCII collating sequence in Table A-3 of Appendix A. The collating sequence can be modified by the PROGRAM COLLATING SEQUENCE clause.

CHARACTER-STRINGS

A character-string is a character or a sequence of contiguous characters that forms a PICTURE character-string, a COBOL word, a literal, or a comment-entry. A character-string is delimited by one of the separators defined above.

Picture-strings

A PICTURE character-string (picture-string) consists of certain combinations of characters in the COBOL character set used as a template. See PICTURE in Chapter 7, for a description of the picture-string and the rules governing its use. A punctuation character that is part of a picture-string is not considered as a punctuation character, but as a symbol in that picture-string.

WORD FORMATION

A COBOL word is a character-string of not more than 30 characters chosen from the following set of 64 characters:

0 through 9 (digits)
A through Z (letters)
a through z (lowercase letters)
- (hyphen)
_ (underscore)

All words except level-numbers, section-names, segment-numbers, and paragraph-names must contain at least one alphabetic character or a hyphen. A word must not begin or end with a hyphen. It is delimited by a space, or by proper punctuation. A word may contain more than one embedded hyphen; consecutive embedded hyphens are also permitted.

Examples of valid words are:

ITEM1
1STITEM
1ST-ITEM
3-5

All words are either reserved words or programmer-defined words.

RESERVED WORDS

A reserved word is one of a specified list of words that may be used in COBOL source programs, but which may not appear as programmer-defined words. They may be used only as specified in the general formats. The types of reserved words are:

- Keywords
- Optional words
- Connectives
- Figurative constants
- Special-character words
- Implementor-names

Prime COBOL 74 reserved words are listed in Table A-2 of Appendix A.

Keywords

A keyword is required when the statement in which the word appears is used in a source program. Within each statement format in this manual, such words are uppercase and underlined.

Optional Words

Within each format, uppercase words that are not underlined are optional words. The presence or absence of an optional word does not alter the meaning of the COBOL program in which it appears, but, when present, the word improves readability of the program.

Connectives

The three types of connectives are:

- Qualifier connectives OF, IN, which are used to associate a data-name, condition-name, text-name, or paragraph-name with its qualifier. Qualifiers are discussed in QUALIFICATION, SUBSCRIPTING, AND INDEXING below.
- Series connectives comma (,) or semicolon (;), which may be used to link two or more consecutive operands.
- Logical connectives AND, OR, which are used in the formation of conditions.

Figurative Constants

Figurative constants are reserved words used to name and reference specific constant values. A figurative constant represents as many instances of the associated character as required in the context of the statement. The singular and plural forms are equivalent and may be used interchangeably.

A figurative constant may be used wherever "literal" appears in a format description; except that, whenever the literal is restricted to numeric characters, the only figurative constant permitted is ZERO (ZEROS, ZEROES). A figurative constant must not be bounded by quotation marks.

Figurative constants are:

Constant	Meaning
ZERO ZEROS ZEROES	The ASCII character represented by hexadecimal B0
LOW-VALUE LOW-VALUES	The character whose hexadecimal representation is 00 (lowest character in the ASCII collating sequence)
HIGH-VALUE HIGH-VALUES	The character whose hexadecimal representation is FF (highest character in the ASCII collating sequence)
QUOTE QUOTES	The quotation mark, whose hexadecimal representation is A2 (")
SPACE SPACES	The blank character represented by hexadecimal A0
ALL literal	Represents one or more of the string of characters comprising the literal. The literal must be either a nonnumeric literal or a figurative constant other than ALL literal. When a figurative constant is used, the word ALL is redundant and is used for readability only.

Special-character Words

The arithmetic operators and relation characters are reserved words. They are:

Operators	Meaning
Arithmetic:	
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
Relation:	
=	Is equal to
<	Is less than
>	Is greater than

Implementor-names

Implementor-names include device-names and switch-names unique to Prime computers. These are listed in Chapter 6, THE ENVIRONMENT DIVISION.

PROGRAMMER-DEFINED WORDS

A programmer-defined word is one supplied by the user to satisfy the format of a clause or statement. Each is constructed according to the rules for word formation. The categories for these words include:

Level-numbers

Data-names

File-names

Condition-names

Mnemonic-names

Paragraph-names

Section-names

Segment-numbers

With the exception of paragraph-names, segment-numbers, and section-names, all programmer-defined words must contain at least one alphabetic character, an underscore, or a hyphen.

If a programmer-defined word is not unique, there must be a unique method of referencing it by using qualifiers (for example, TAX-RATE IN STATE-TABLE). Qualifiers are explained in QUALIFICATION, SUBSCRIPTING, AND INDEXING below.

Level-numbers

Level-numbers are one- or two-digit, programmer-defined numbers in the DATA division. They group items within the data hierarchy of the Record Description.

The range of levels is 01 through 49, and 66, 77, and 88. Level-numbers 1 through 9 may be written as single digits. The use of level numbers is discussed in Chapter 7, THE DATA DIVISION.

Data-names

A data-name is a word made up by the user to identify a data item used in a program. A data-name always refers to a field of data, not to a particular value. It is formulated according to the rules for word formation above. It must not be identical to a reserved word.

Data-names are used in all divisions of a COBOL program. When referenced in the PROCEDURE division, a data-name, if not unique, must be followed by a syntactically correct combination of qualifiers, subscripts, or indexes necessary to ensure uniqueness.

File-names

A file is a collection of data records of a similar class or application. A file-name is preceded by an FD entry in the DATA division's FILE SECTION. Rules for composition of the name are identical to those for data-names. (See WORD FORMATION above.) References to a file-name appear in PROCEDURE division I-O statements as well as in the ENVIRONMENT and DATA divisions.

Condition-names

A condition-name is a name assigned to a specific value, set of values, or range of values, within a complete set of values that a data item may assume. The data item is called a conditional variable. Condition-names are allowed in the FILE, WORKING-STORAGE, and LINKAGE sections of the DATA division, as well as in the SPECIAL-NAMES paragraph of the ENVIRONMENT division.

A condition-name is defined within the DATA division in a level-88 entry subordinate to the associated data item name, or in the SPECIAL-NAMES paragraph, assigned to the ON STATUS or OFF STATUS of switches. Rules for the formation of condition-name words are the same as those specified in WORD FORMATION. Additional information concerning condition-names and procedural statements employing them is given in the chapters on the DATA and PROCEDURE divisions.

Mnemonic-names

A mnemonic-name is assigned in the ENVIRONMENT division in the SPECIAL-NAMES paragraph for reference in ACCEPT or DISPLAY statements or in switch-condition tests. A mnemonic-name is composed according to the rules for word formation above.

Paragraph-names and Section-names

These are words that identify paragraphs and sections, respectively, in the PROCEDURE division. They may be up to 30 characters long, and may be all alphabetic, all numeric, or alphanumeric.

Examples of valid paragraph-names are:

```
050-NEXT-ITEM
050
NEXT-ITEM
```

Segment-numbers

A segment-number must be an integer between 0 and 99.

LITERALS

A literal is a programmer-defined constant value. It is not identified by a data-name in a program, but is completely defined by its own identity. A literal is either nonnumeric or numeric.

Nonnumeric Literals

A nonnumeric literal must be delimited by matching quotation marks or apostrophes. (Note that double quote marks may be used in entering a COBOL program with the EDITOR only if the user has changed the erase character from quote marks to some other character with the PRIMOS command TERM or the EDITOR command SYMBOL.) All spaces enclosed by the delimiters are included as part of the literal. The length of a nonnumeric literal is computed excluding the delimiters. A nonnumeric literal must not exceed 120 characters in length. Minimum length is 1.

A nonnumeric literal may be any combination of characters in the ASCII set.

If the delimiter itself is to be used within the literal string, it should be written twice. The last example below shows a single quote within a literal delimited by single quotes.

The following are examples of nonnumeric literals:

```
"ILLEGAL CONTROL CARD"

'IT WAS A DARK AND STORMY NIGHT'

"123"
```

```
'1001'

"3.1414"

'-6'

"DO'S AND DON'TS"

'HERE''S LOOKING AT YOU'
```

Numeric Literals

A numeric literal must contain at least one and not more than 18 digits. A numeric literal may consist of the characters (digits) 0 through 9 (optionally preceded by a sign) and a decimal point, or a comma in the case DECIMAL-POINT IS COMMA discussed below. It may contain only one sign character and only one decimal point. The sign, if present, must appear as the leftmost character of the numeric literal. If a numeric literal is unsigned, it is assumed to be positive.

A decimal point may appear anywhere within the numeric literal, except as the rightmost character. If a numeric literal does not contain a decimal point, it is considered to be an integer.

If a literal conforms to the rules for the formation of numeric literals, but is enclosed in quotation marks, it is a nonnumeric literal and it is treated as such by the compiler.

The following are examples of numeric literals:

```
72    +1011    3.14159    -6    -.333    1.23E10
```

The last example uses floating-point format, a Prime extension that is described in the section on DATA REPRESENTATION AND ALIGNMENT in this chapter.

By use of the clause DECIMAL-POINT IS COMMA, the functions of the period and comma characters may be interchanged, putting the European notation into effect. In this case, for example, the literal value one thousand and one tenth would be written as 1.000,1.

Continuation of Literals

When a literal is too long to fit on one line, the following conventions apply to the next line of coding (continuation line):

- A hyphen in the indicator area (column 7) of a line indicates that the first nonblank character in area B of the current line is the successor of the last nonblank character of the preceding line without any intervening space.

- If the continued line contains a nonnumeric literal without closing quotation mark, the first nonblank character in area B on the continuation line must be a quotation mark, and the continuation starts with the character immediately after that quotation mark. All spaces at the end of the continued line are considered part of the literal.

- Area A of a continuation line must be blank.

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

  The next two lines illustrate continuation of a nonnumeric literal.

```
    MOVE 'NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF
-   'THE PARTY.' TO HEADER.
```

DATA LEVELS

The two levels of data are group and elementary.

Group Item

A group item is defined as one having further subdivisions, so that it contains one or more elementary items or other groups. The maximum size of a group item is given in Appendix J.

Elementary Item

An elementary item is a data item containing no subordinate items. An elementary item must contain a PICTURE clause, except when usage is described as COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, or INDEX. The maximum size of an elementary item is given in Appendix J.

CLASSES AND CATEGORIES OF DATA

The classes of data are: alphabetic, numeric, and alphanumeric. Within these, the categories of data are: alphabetic, numeric, numeric edited, alphanumeric, and alphanumeric edited. Every elementary item except an index data item belongs to a class and to a category, as defined by its PICTURE or USAGE clause.

The category of a data item is used to determine the validity of operations such as MOVE and COMPUTE, and for alignment. The categories have the following characteristics (more detail is given under the PICTURE clause in Chapter 7).

Alphabetic Item

An alphabetic item consists of any combination of the 26 uppercase characters of the English alphabet and the space character. It is defined by PICTURE A.

Numeric Item

A numeric item consists only of digits and no more than one assumed decimal point and an optional sign. It is defined by PICTURE 9 or by USAGE IS COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, or COMPUTATIONAL-3.

Numeric Edited Item

An edited numeric item contains only digits and special editing characters or editing characters alone, as described under PICTURE in Chapter 7. It is defined by PICTURE 9 plus editing characters.

Alphanumeric Item

An alphanumeric item consists of any combination of ANSI characters plus lowercase letters, defined by PICTURE X, A, or 9.

Alphanumeric Edited Item

This is an alphanumeric item defined by PICTURE X or PICTURE A plus editing characters described under PICTURE in Chapter 7.

Relationship of Classes and Categories

The class rather than the category is used in some relation conditions, and for determining validity of operations on group items. For alphabetic and numeric elementary items, classes and categories are the same. For elementary items, the alphanumeric class includes the categories of alphanumeric edited, numeric edited, and alphanumeric. The class of a group item is treated at execution time as alphanumeric regardless of the class of elementary items subordinate to that group item. Table 4-2 depicts the relationship of the class and categories of data items.

Table 4-2
Classes and Categories of DATA

Level of Data	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric Edited
		Alphanumeric Edited
		Alphanumeric
Group	Alphanumeric	Alphabetic
		Numeric
		Numeric Edited
		Alphanumeric Edited
		Alphanumeric

DATA REPRESENTATION AND ALIGNMENT

Data is further categorized by the format in which it is stored in the computer. The formats are: display or unpacked decimal, packed decimal, binary, index, and floating-point. These formats are specified by the USAGE clause, as outlined below.

USAGE	Machine Description
DISPLAY	Unpacked decimal
COMPUTATIONAL-3	Packed decimal
COMPUTATIONAL	Binary
INDEX	Binary
COMPUTATIONAL-1	Single-precision floating-point
COMPUTATIONAL-2	Double-precision floating-point

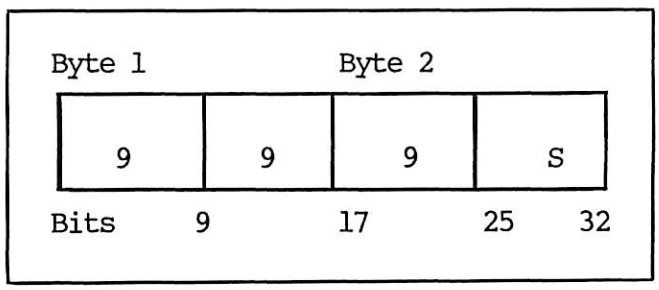
Note

Data items of all formats may be used together in computations, although time is often saved by assuring that all data items used in any given computation are in the same format.

COBOL operates on five types of decimal data: leading separate sign, trailing separate sign, packed decimal, leading embedded sign, and trailing embedded sign. The last two types may be entered with an overpunch. Table A-10 in Appendix A summarizes the characteristics of each decimal data type and the sign values.

Unpacked Decimal Item (DISPLAY)

An unpacked decimal item is one in which one byte (eight binary bits) is employed to represent one digit as well as the sign. An exception is such an item with SIGN IS SEPARATE clause, discussed in Chapter 7. The PICTURE clause for an external decimal item may contain only 9, S, V, and P. The USAGE for an unpacked decimal item is always DISPLAY, whether implicit or explicit. Maximum size is 18 digits. The storage of such an item may be represented in Figure 4-2.



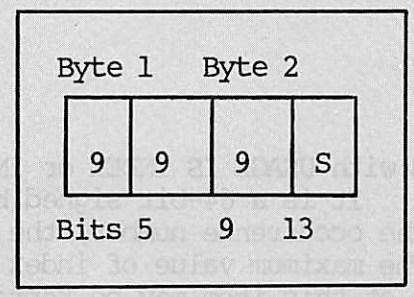
Unpacked Decimal Storage
Figure 4-2

Packed Decimal Item (COMP-3)

A packed decimal item is one in which each byte represents two digits. It is defined by the COMPUTATIONAL-3 or COMP-3 USAGE clause. Maximum size is 18 digits.

Its PIC may contain only 9, S, V, P. A packed decimal item defined by n nines in its PICTURE occupies (n/2)+1 bytes in memory. All bytes, except the rightmost, contain a pair of digits.

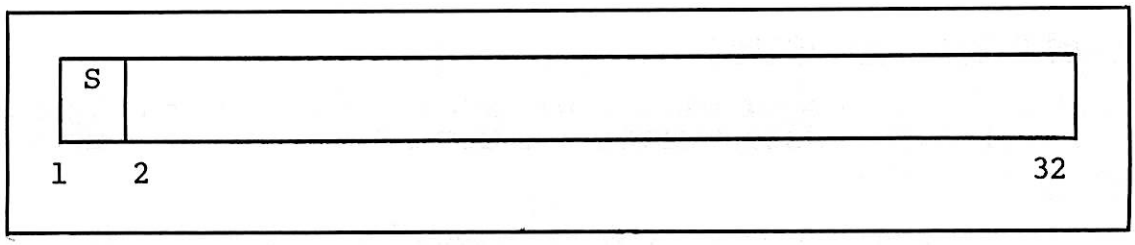
The rightmost half-byte of a packed item contains a representation of the sign. Bit string 1100 represents a positive sign, 1101 represents a negative sign. Four bits are always reserved for the sign in a packed field, even if the picture lacks the leading character S. For this reason, the optimal space allocation for a packed decimal item is an odd-size field. The storage of such an item may be represented in Figure 4-3.



Packed Decimal Storage
Figure 4-3

Binary Item (COMP)

A binary item uses the base-2 system to represent an integer. The item occupies the following storage: 16 bits if up to 4 nines are specified in the PICTURE clause, 32 bits if 5-9 nines are specified and 64 bits if 10-18 nines are specified. The maximum size is 18 digits. If no PICTURE is specified, the default is 16 bits. The leftmost bit of the storage area is the operational sign: 0 is positive, 1 is negative. (COMPUTATIONAL data types are stored in two's-complement form.) The sign is optional in the PICTURE clause. If it is omitted, the value in this field is always treated as positive. USAGE IS COMPUTATIONAL must be specified. Since this data type is represented in hardware by a signed data type (fixed binary), extra code is required to return the absolute value for every reference to this field when it is declared without a sign. In addition, if the PICTURE clause of COMP items specifies more than nine digit positions, or specifies positions to the right of the decimal point, extra code is required to convert the contents of such fields when they are referenced. The storage of this item may be represented in Figure 4-4.

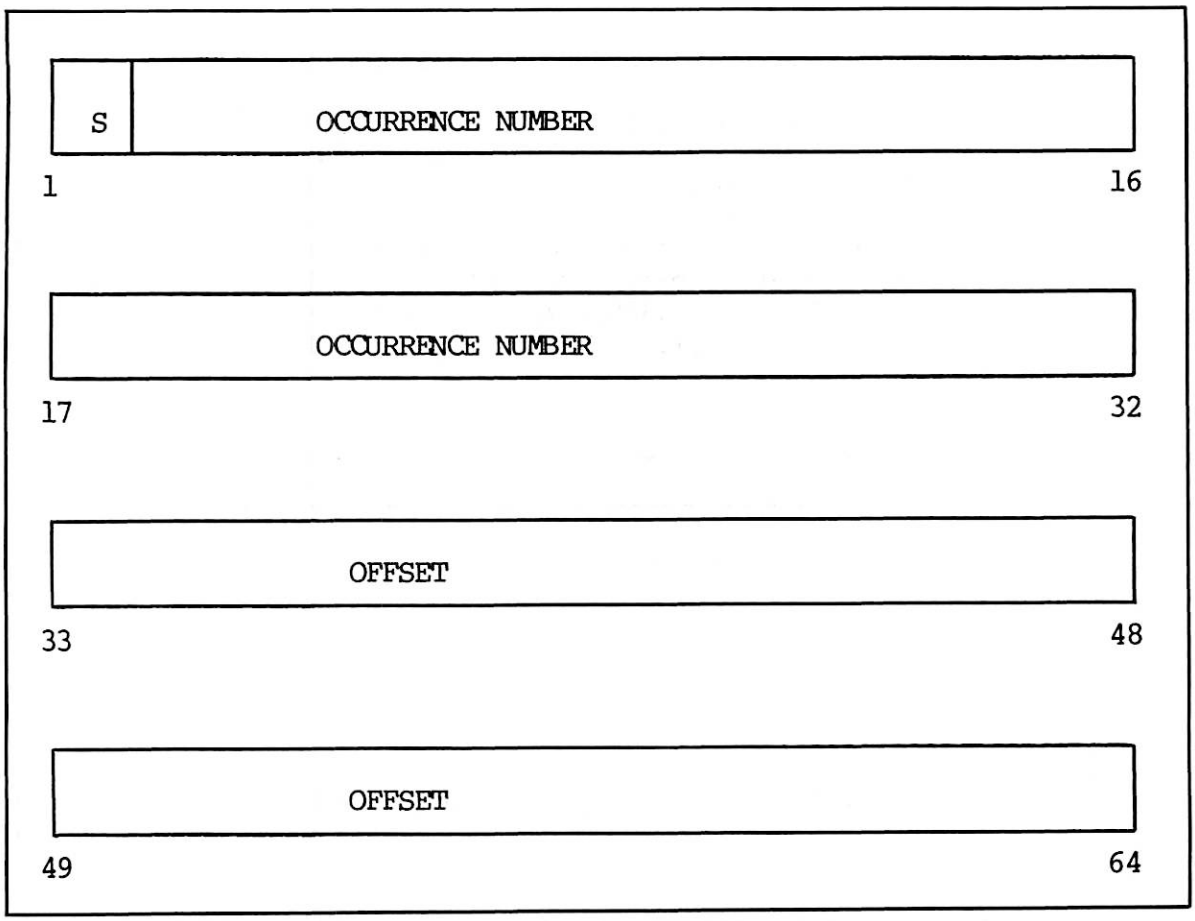


Binary Storage (PIC S9(9))
Figure 4-4

COMP items are aligned by the compiler on halfword (16-bit) boundaries. Sixteen-bit binary items have a range of -32768 to +32767. Thirty-two-bit binary items have a range of -2147483648 to +2147483647. (These two items correspond to INTEGER*2 and INTEGER*4, respectively, in FORTRAN.) Sixty-four-bit binary items are converted to decimal when referenced and therefore have the same range as 18-digit decimal data, that is, a PIC of from 9(18) to V9(18).

Index Item

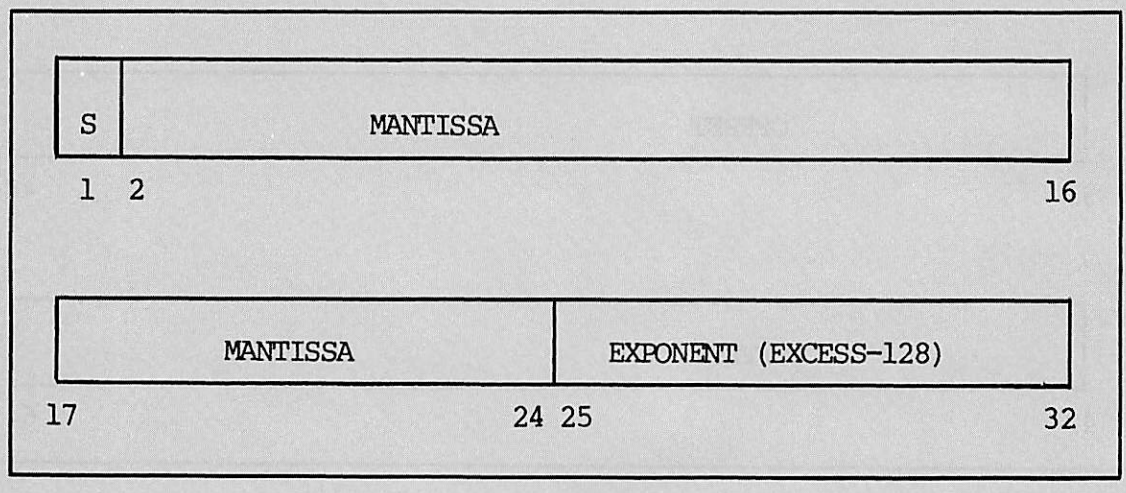
An index item is defined with USAGE IS INDEX or INDEXED BY. It may not have a PICTURE clause. It is a 64-bit signed binary item, the first half of which contains the occurrence number, the last half of which contains the offset. The maximum value of index items is discussed in Appendix J. The storage of this item may be represented by Figure 4-5.



Index Storage
Figure 4-5

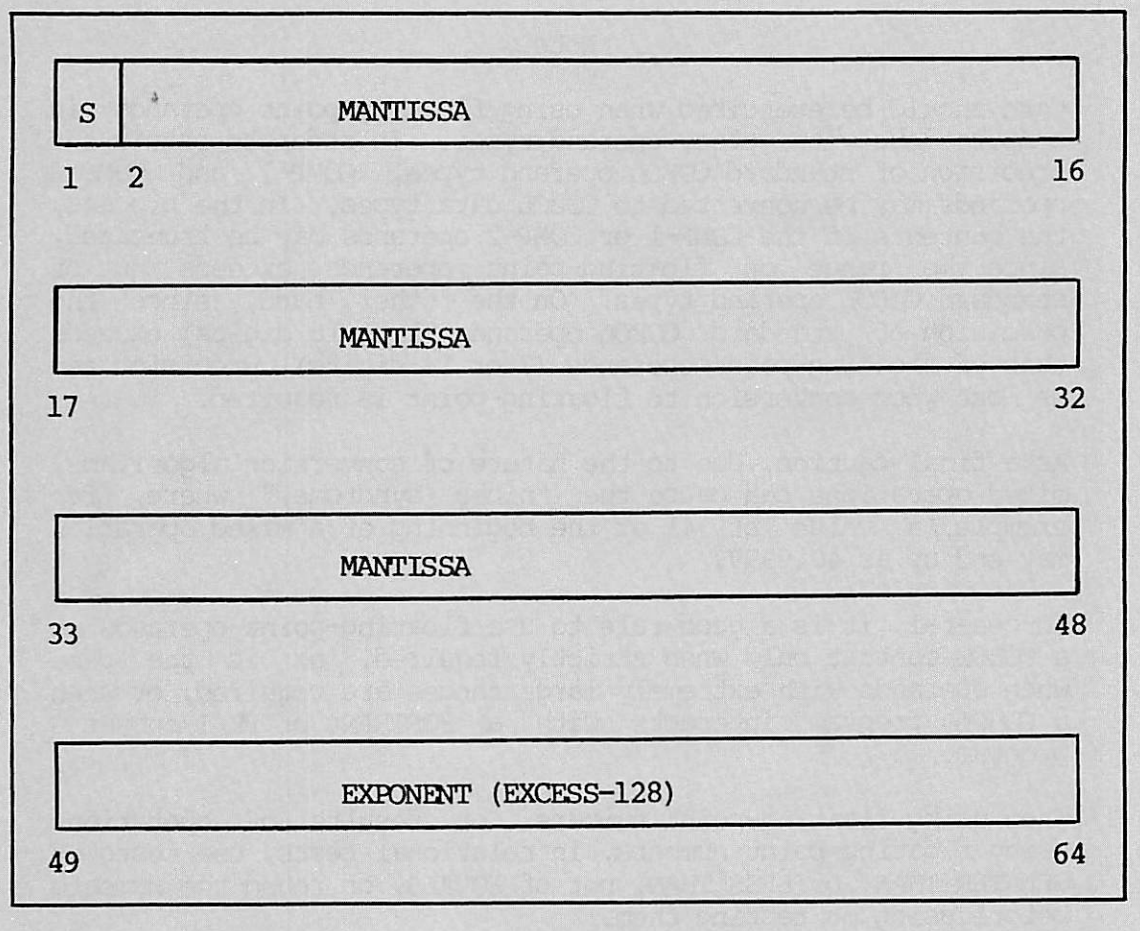
Floating-point Item: Prime Extension (COMP-1, COMP-2)

A single-precision floating-point item is defined by a USAGE clause of COMPUTATIONAL-1 or COMP-1. No PICTURE clause is allowed. The item occupies 32 bits of which bit 1 (the leftmost bit) is the sign, bits 2-24 are the mantissa, and bits 25-32 contain the exponent. The sign and mantissa are treated as a two's-complement number, and the exponent is an unsigned excess-128 binary exponent. Effective precision is between 22 and 23 bits (± 8,388,607). The exponent range is -128 to +127 (10 to the ±38 power). The storage of this item may be represented in Figure 4-6.



Single-precision Floating-point Storage
Figure 4-6

A double-precision floating-point item is defined by a USAGE clause of COMPUTATIONAL-2 or COMP-2. No PICTURE clause is allowed. The item occupies 64 bits of which bit 1 (the leftmost bit) is the sign, bits 2-48 are the mantissa, and bits 49-64 contain the exponent. The sign and mantissa are treated as a two's complement number, and the exponent is an unsigned excess-128 binary exponent. Effective precision is between 46 and 47 bits (± 737,488,355,327). The exponent range is -32896 to +32639 (10 to the +9823 or -9812 power). The storage of this item may be represented in Figure 4-7.



Double-precision Floating-point Storage
Figure 4-7

Floating-point format is a Prime extension to ANSI COBOL intended for use in scientific calculations, when very large or very small numbers must be represented, or when the user wishes to call FORTRAN or PL1G subroutines that operate on floating-point (real) numbers.

In a COBOL statement, the format of a floating-point number is:

[(±)]mantissaE[(±)]exponent

The mantissa consists of one to seven digits for COMP-1 or one to 14 characters for COMP-2 with a required decimal point. Examples are:

```
MOVE 1.23456E-10 TO ITEM1.
IF TEST1 > 4.0E14 PERFORM 050-EXCESS.
```

Floating-point items are compiler-aligned on halfword (16-bit) boundaries.

Note

Care should be exercised when using floating-point operands in computations with other operand types. In order to retain the precision of standard COBOL operand types, COMP-1 and COMP-2 operands may be converted to COBOL data types. In the process, the contents of the COMP-1 or COMP-2 operands may be truncated, since the range of floating-point operands exceeds that of standard COBOL operand types. On the other hand, since the precision of standard COBOL operands (1 to 18 digits) exceeds that of floating-point operands (7 or 14 digits), precision can be lost when conversion to floating-point is required.

As a final caution, due to the nature of conversion algorithms, mixed operations can cause the "nines syndrome," where, for example, a value of 41 at the beginning of a mixed operation may end up as 40.9999.

In general, it is a good rule to use floating-point operands in a COBOL context only when strictly required, as is the case when operands with extremely large ranges are required, or when a COBOL program interacts with a FORTRAN or PL/I subset G program.

When using floating-point numbers, or results of operations using floating-point numbers, in relational tests, use tests of GREATER THAN or LESS THAN, not of EQUALS, or round the numbers before using or testing them.

Standard Alignment Rules

The Prime COBOL compiler automatically aligns data as needed at compilation time. DISPLAY and COMP-3 items are aligned on byte boundaries, with the exceptions discussed in the next section. All other items are aligned on 16-bit boundaries. At execution time, the standard rules by which the compiler positions data within an elementary item depend on the category of the receiving item. These rules are:

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

   - The data is aligned by decimal point and is moved to the receiving digit positions with zero filling or truncation at either end, as required.

   - When an assumed decimal point is not explicitly specified, the data item is treated as if it had an assumed decimal point immediately following its rightmost digit. It is aligned as in the rule above.

2. If the receiving data item is numeric edited, the data moved to the edited data item is aligned by decimal point. Zero filling or truncation at either end occurs as required except where editing requirements cause replacement of the leading zeros.

3. If the receiving data item is alphanumeric (other than a numeric edited data item), alphanumeric edited, or alphabetic, the sending data is aligned at the leftmost character position in the receiving data item. Space filling or truncation occurs to the right, as required.

If the JUSTIFIED clause is specified for the receiving item, these standard rules are modified as described under JUSTIFIED in Chapter 7.

The alignment examples below show the results of moving various length alphabetic and alphanumeric items into an 11-character alphanumeric field. (b = blank)

Data to Be Stored	Receiving Field Before Transfer	Receiving Field After Transfer
ABC	XXXXXXXXXXX	ABCbbbbbbbb
ABCDEF1234	XXXXXXXXXXX	ABCDEF1234b
AAABBBCCCDD	XXXXXXXXXXX	AAABBBCCCDD
AAABBBCCCDDDE	XXXXXXXXXXX	AAABBBCCCDD

The examples below show the results of moving various length numeric items into a six-character numeric field. (^ = implied decimal point.)

Data to Be Stored	Receiving Field Before Transfer	Receiving Field After Transfer
3^4	999V999	003^400
345^678	999V999	345^678
12345^67890	999V999	345^678
34^	999V999	034^000
1234567890	999V999	890^000
1234567890	9999V99	7890^00
3^4	999999	000003

Alignment of Substructures Within Structures -- Prime Extension

The compiler automatically aligns certain elements on 16-bit boundaries in order to allow substructures to be passed to called programs correctly. Alignment follows these rules:

1. Each level-01 or level-77 item is allocated on a 16-bit boundary.

2. Each group item subordinate to a level-01 item is aligned on the largest boundary required by any item contained in it:

COBOL Type	Alignment Required
DISPLAY	Byte
COMP	16 bits
COMP-1	16 bits
COMP-2	16 bits
COMP-3	Byte

3. Compiler-generated filler is inserted into structures where necessary to make substructures align on the proper boundary.

4. If the -SLACKBYTES option is specified at compile time, a diagnostic is issued by the compiler when filler is added to align a substructure. If the -MAP option is specified, each data item so aligned is indicated by the phrase COMPILER-ALIGNED.

Examples: The following structure is to be passed to a called program. A and B may be byte-aligned, while C requires 16-bit alignment:

```
01 STRUC1.
    02 A            PIC X.
    02 S2.
       03 B         PIC X.
       03 C         COMP.
```

The compiler actually allocates the structure as:

```
01 STRUC1.
    02 A            PIC X.
    02 FILLER       PIC X.
    02 S2.
       03 B         PIC X.
       03 C         COMP.
```

When S2 is passed to a called program, this automatic alignment allows the programmer to pass the subgroup (a Prime extension) because it is already aligned to correspond to a level-01 or level-77 group in the called program. Thus the argument can be described as a level-01 group in the called program:

```
LINKAGE SECTION.
01 M.
    02 B PIC X.
    02 C COMP.
  .
  .
PROCEDURE DIVISION USING M.
  .
  .
```

ALGEBRAIC SIGNS

Algebraic signs fall into two categories: operational signs and editing signs. Operational signs are associated with signed numeric data items and signed numeric literals to indicate their algebraic properties. Editing signs appear on edited reports to identify the sign of the item.

The SIGN clause permits the programmer to state explicitly the location of the operational sign. Editing signs are inserted into a data item with the editing symbols of the PICTURE clause.

QUALIFICATION, SUBSCRIPTING, AND INDEXING

Qualification of Names

The user must be able to identify, uniquely, every name that defines an element in a COBOL source program. The name may be unique in its spelling or hyphenation, or unique reference may be accomplished by use of qualifier names.

Qualifiers are names of higher-level items (that is, of a lower level-number) preceded by the word OF or IN. A series of items connected by OFs or INs may qualify one name. The general formats for qualification are:

Format 1

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

Format 2

paragraph-name [{ **OF** / **IN** } **section-name**]

Format 3

'file-name' [{ **OF** / **IN** } **'UFD-name'**]

Format 4

status-name OF switch-name

The rules for qualification are:

- Each qualifier must be of a higher level and within the same hierarchy as the name it follows.
- The same name must not appear at two levels in the same hierarchy.
- If a data-name or a condition-name is assigned to more than one item in a source program, the name must be qualified each time it is referred to.
- Paragraph-names may be qualified only by their section-name. Therefore, two identical paragraph-names should not appear in the same section. When a paragraph is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to within the same section.
- A name can be qualified even though it does not need qualification. If more than one combination of qualifiers can make a name unique, any combination can be used. The complete set of qualifiers for a data-name must not be the same as any partial set of qualifiers for another data-name.
- The maximum number of qualifiers is one for a paragraph-name and 50 for a data-name or condition-name. File-names may be qualified only in a COPY statement. Mnemonic-names and section-names must not be qualified.

In the following example, the data-name YEAR will require qualification for reference because it defines two elementary items, one in HIRE-DATE and one in TERMINATION-DATE.

```
01  EMPLOYEE-RECORD
    05  NAME
    05  ADDRESS
    05  HIRE-DATE
        10  YEAR
        10  MONTH
        10  DAYY
    05  TERMINATION-DATE
        10  YEAR
        10  MONTH
        10  DAYY
```

YEAR OF HIRE-DATE is a qualified reference that differentiates between year fields in HIRE-DATE and TERMINATION-DATE. YEAR OF HIRE-DATE IN EMPLOYEE-RECORD is also a valid qualifier for the first YEAR field.

Subscripting

Subscripts can be used only when reference is made to an individual element within a list or table of like elements that have not been assigned individual data-names. (See the OCCURS clause in Chapters 7 and 10.)

The format for subscripting is:

{ data-name / condition-name } (subscript-1 [, subscript-2 [, subscript-n] ···])

Subscripting is discussed in more detail in Chapter 10.

The subscript can be represented by a numeric literal, by a numeric data-name, or by an arithmetic expression. The data-name subscripts may themselves be qualified or subscripted.

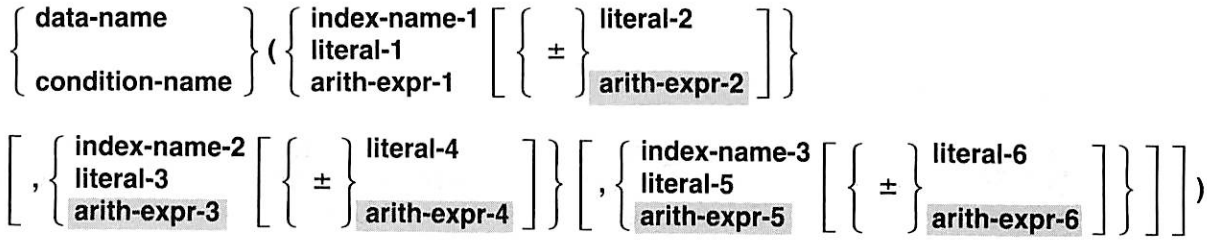
The subscript may be signed. It must have a positive integer value. The lowest possible subscript value is 1. This value points to the first element of the table. The next elements of the table are pointed to in turn by subscripts whose values are 2, 3, and so on. The highest permissible subscript value, in any particular case, is the maximum number of occurrences of the item as specified in the OCCURS clause. Literal subscripts are range-checked at compile time. Variable subscripts can be checked at runtime if the -RANGE option is specified at compile time.

The subscript that identifies the table element is delimited by the balanced pair of separators, left parenthesis and right parenthesis, following the table element data-name. When more than one subscript is required, they are written in the order of successively less inclusive dimensions of the table organization.

Indexing

References can be made to individual elements within a table of like elements by specifying indexing for that reference. Indexing differs from subscripting because it uses as a pointer an item that may refer to one table only and thus may be used implicitly by some statements. An index is assigned to a level of a table by using the INDEXED BY phrase in the definition of the table. A name given in the INDEXED BY phrase is known as an index-name and is used to refer to the assigned index. The value of an index corresponds to the occurrence number of an element in the associated table. An index-name must be initialized before it is used in a table reference. An index-name can be given an initial value by a SET, a SEARCH ALL, or a Format-4 PERFORM statement.

The general format for indexing is:



Prime COBOL supports two types of indexing: direct and relative. Direct indexing is specified by using an index-name in the form of a subscript. Relative indexing uses a computed index value. It is specified when the index-name is followed by a space, one of the operators + or -, another space, and an unsigned integer numeric literal, all delimited by left parenthesis and right parenthesis following the table name. An example is:

```
TABLE(INDEX-NAME + 1)
```

The occurrence number resulting from relative indexing is determined by incrementing or decrementing the index by the value of the literal.

When more than one index-name is required, they are written in the order of successively less inclusive dimensions of the data organization.

When a statement that refers to an indexed table element is executed, the value in the associated index must be neither less than 1, nor greater than the limit specified by the OCCURS clause. This restriction also applies to the values resulting from relative indexing.

Indexing is discussed in more detail in Chapter 10.

Restrictions on Qualification, Subscripting, and Indexing

- Indexing is not permitted where subscripting is not permitted.
- An index may be modified only by the SET, SEARCH, and Format-4 PERFORM statements.
- Data items described by the USAGE IS INDEX clause permit storage of the values associated with index-names. Such data items are called index data items.

ARITHMETIC EXPRESSIONS

Definition

An arithmetic expression must be one of the following:

- A name of a numeric elementary item
- A numeric literal
- Such names and literals separated by arithmetic operators
- Two arithmetic expressions separated by an arithmetic operator
- An arithmetic expression enclosed in parentheses

Any arithmetic expression may be preceded by a unary operator. The permissible combinations of variables, numeric literals, arithmetic operators, and parentheses are given in Table 4-3.

Table 4-3
Symbol Combinations in Arithmetic Expressions

(P = Permitted, X = Invalid)

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

Names and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic may be performed.

Arithmetic Operators

The characters below represent the binary and unary arithmetic operators. They must be preceded by at least one space.

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

Unary Arithmetic Operators	Meaning
+	The effect of multiplication by +1 (sign normalization).
-	The effect of multiplication by -1 (sign inversion).

Parentheses	Meaning
( )	Used to enclose expressions to control the sequence in which conditions are evaluated.

Rules

Follow these general rules for arithmetic expressions:

- Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first. Within nested parentheses, evaluation proceeds from the least inclusive set to the most inclusive set.
- When parentheses are not used, the following hierarchical order of execution is implied:
    1. Unary plus and minus
    2. Exponentiation

3. Multiplication and division

4. Addition and subtraction

The order of execution of consecutive operations of the same hierarchical level is from left to right. Example:

```
A + B / ( C - D * E )
```

This expression is evaluated in the following sequence:

1. Compute the product D times E, considered as intermediate result R1.

2. Compute intermediate result R2 as the difference C - R1.

3. Divide B by R2, providing intermediate result R3.

4. Compute the final result by addition of A to R3.

Without parentheses, the expression A + B / C - D * E is evaluated as:

```
R1 = B/C
R2 = D * E
R3 = A + R1
```

The final result is R3 - R2.

- When parentheses are employed, the following punctuation rules should be used:

  1. A left parenthesis is preceded by one or more spaces.

  2. A right parenthesis is followed by one or more spaces.

- Operators, variables, and parentheses may be combined in arithmetic expressions as summarized in Table 4-3.

- An arithmetic expression may begin only with one of the symbols (, +, -, or a variable; it may end only with a) or a variable. Each left parenthesis is to the left of its corresponding right parenthesis.

Arithmetic Statements

The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements. These have several common features.

- The data descriptions of the operands need not be the same; any necessary conversion and decimal point alignment is supplied throughout the calculation.

- The maximum size of each operand is 18 decimal digits. The composite of operands, which is a hypothetical data item resulting from the superimposition of operands aligned on their decimal points, must not contain more than 18 decimal digits. An example is given in Chapter 8 in the section on Arithmetic Statements in the PROCEDURE Division.

Overlapping Operands

When a sending and a receiving item in an arithmetic statement or an INSPECT, MOVE, SET, STRING, UNSTRING, or other statements share a part of their storage areas, the result of the execution of such a statement is undefined and unpredictable.

CONDITIONAL EXPRESSIONS

Definition

Conditional expressions identify conditions that are tested to enable the object program to select between alternate paths of control depending upon the truth value of the condition.

Simple Conditions

The simple conditions are the relation, class, condition-name, switch-status, and sign conditions. A simple condition has a truth value of "true" or "false." The inclusion in parentheses of simple conditions does not change the simple truth value.

Relation Condition: A relation condition causes a comparison of two operands. A relation condition has these formats:

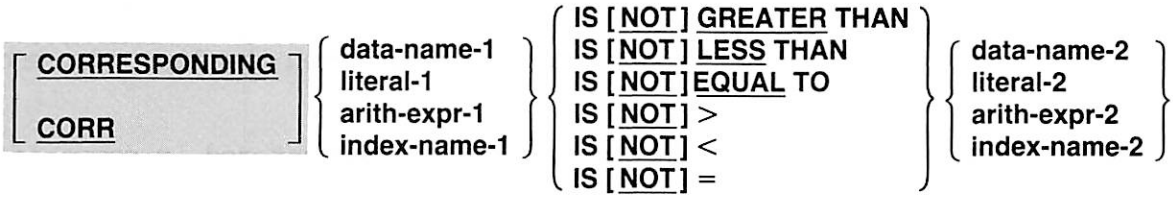
Format 1: **operand relation operand**

Format 2: [**CORRESPONDING** / **CORR**] **operand relation operand**

The relation is a relational operator: equals, greater, less, or the negation of one of these. A relation condition has a truth value of "true" if the relation exists between the operands. The operand is a data-name, literal, arithmetic expression, or figurative-constant.

The CORRESPONDING option in relation conditions is a Prime extension and can be used only when the operands are group items.

The general format of a relation condition is as follows:



Note

Although required where indicated in formats, the relational characters <, >, and = are not underlined in this text.

The relational operator specifies the type of comparison to be made in a relation condition. A space must precede and follow each reserved word comprising the relational operator. When used, NOT and the next keyword or relation character form one relational operator defining the comparison to be executed for truth value. Thus NOT EQUAL is a truth test for an unequal comparison; NOT GREATER is a truth test for an equal or less comparison.

Comparison of two numeric operands of different formats is permitted. If either operand is nonnumeric, the comparison is nonnumeric.

- Numeric comparisons: For elementary operands whose class is numeric, a comparison is made with respect to their algebraic value. The length of the operands is not significant. Zero is considered a unique value regardless of the sign. It is neither positive nor negative, and will fail these sign tests.

  Comparison of these operands is permitted regardless of their usage. Unsigned numeric operands are considered positive.

  The data operands are compared after alignment of their decimal points. An index-name or index item may appear in a numeric comparison.

- Nonnumeric comparisons: For nonnumeric operands, a comparison is made with respect to the Prime collating sequence of characters. The value associated with each ASCII character in the Prime computer is the basis for the comparison. (Refer to Appendix A for all ASCII character representations and the Prime collating sequence.)

Comparison proceeds by comparing characters in corresponding character positions starting from the high-order (left) end and continuing until either a pair of unequal characters is encountered or the low-order end of the operand is reached. The operands are determined to be equal if all pairs of characters compare equally through the last pair, when the low-order end is reached.

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

The size of an operand is the total number of characters in the operand.

If the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by sufficient spaces to make the operands of equal size.

If one operand is a literal, the data class of the two operands must be the same.

If one of the operands is specified as numeric, it must be an integer data item or an integer literal and:

- If the nonnumeric operand is an elementary data item or a nonnumeric literal, the numeric operand is treated as though it were an elementary alphanumeric data item of the same size as the numeric data item.
- If the nonnumeric operand is a group item, the numeric operand is treated as though it were a nonnumeric item of the same size as the numeric data item.
- A noninteger numeric operand cannot be compared to a nonnumeric operand.
- Numeric and nonnumeric operands may be compared only when their usage is the same.

Class Condition: The class condition determines whether the contents of a data-name are numeric or alphabetic. A numeric data item consists entirely of the digits 0 through 9, with or without the operational sign. An alphabetic data item consists entirely of the uppercase alphabetic characters and the space. The general format for the class conditions is:

data-name IS [NOT] { **NUMERIC** / **ALPHABETIC** }

The data-name must be described, implicitly or explicitly, as USAGE IS DISPLAY or USAGE IS COMPUTATIONAL-3.

The NUMERIC test cannot be used with a data-name described as alphabetic or as a group item composed of elementary items whose data description indicates the presence of operational signs.

If the data description of the data-name being tested does not contain an operational sign, the data-name is determined to be numeric only if the contents are numeric and an operational sign is not present.

If the data description of the data-name being tested does contain an operational sign, the data-name is determined to be numeric only if the contents are numeric and a valid operational sign is present.

The ALPHABETIC test cannot be used with a data-name described as numeric. The data-name being tested is determined to be alphabetic only if the contents consist of any combination of the uppercase alphabetic characters and the space.

Condition-name Condition: In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to a value associated with one of its condition-names in a level-88 entry of the DATA division. The general format for the condition-name statement is as follows.

[NOT] condition-name

If the condition-name is associated with a range or ranges of values, then the conditional variable is tested to determine whether or not its value falls in this range, including the end values. (See Chapter 7 for details.)

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

The result of the test is true if the content of the field associated with the condition-name equals one of the values specified for that condition-name.

Switch-status Condition: A switch-status condition determines the ON or OFF status of a switch. The switch-name and the ON or OFF value associated with the condition must be named in the SPECIAL-NAMES paragraph of the ENVIRONMENT division. The general format for the switch-status statement is as follows:

[NOT] status-name

The result of the test is true if the switch is set to the ON or OFF status associated with the switch in the SPECIAL-NAMES paragraph.

Sign Condition: The sign condition determines whether or not the algebraic value of an arithmetic expression is less than, greater than, or equal to 0. The general format for a sign condition is as follows:

```
{ data-name  }
{            } IS [NOT] { POSITIVE }
{ arith-expr }          { NEGATIVE }
                        { ZERO     }
```

Complex Conditions

A complex condition is formed by combining simple conditions, combined conditions and/or complex conditions with logical connectors (logical operators AND and OR), or by negating these conditions with logical negation (the logical operator NOT). The truth of a complex condition is calculated as described in Condition Evaluation Rules below. The logical operators are the following.

Logical Operator	Meaning
AND	Logical conjunction; the truth value is "true" if both of the conjoined conditions are true; "false" if one or both of the conjoined conditions is false.
OR	Logical inclusive OR; the truth value is "true" if one or both of the included conditions is true; "false" if both included conditions are false.
NOT	Logical negation is the reversal of the truth value; the truth value is "true" if the condition is false, and "false" if the condition is true.

Logical operators must be preceded and followed by a space.

Negated Simple Conditions: The general format of a negated simple condition is:

```
NOT simple-condition
```

Thus, the simple-condition is negated through the use of the logical operator NOT.

The truth value of a negated simple condition is the opposite of the truth value for a simple condition. The negated condition is true if the simple condition is false, and false if the simple condition is true.

Inclusion in parentheses of a negated simple condition does not affect the truth value.

Combined and Negated Combined Conditions: Combined conditions are simple conditions connected by one of the logical operators AND or OR. A combined condition has the format:

[NOT] condition-1 { { **AND** / **OR** } **[NOT] condition-2** } ...

where condition is:

- A simple condition
- A negated simple condition
- A combined condition
- A negated combined condition, that is, the logical operator NOT followed by a combined condition enclosed in parentheses
- Combinations of the above

Table 4-4 below sets forth the permissible combinations of conditions, logical operators, and parentheses.

Table 4-4

Permissible Combinations of Conditions, Logical Operations, and Parentheses

Element	Place in Expression First	Last	When not first, the element can be immediately preceded only by:	When not last, the element can be immediately followed only by:
Simple-condition	Yes	Yes	OR, NOT, AND, (	OR, AND,)
OR and AND	No	No	Simple-condition,)	Simple-condition, NOT, (
NOT	Yes	No	OR, AND, (	Simple-condition, (
(	Yes	No	OR, NOT, AND, (	Simple-condition, NOT, (
)	No	Yes	Simple-condition,)	OR, AND,)

Multiple Conditions: Multiple conditions refer to complex conditions grouped in parentheses.

Parentheses are permitted to an arbitrary depth. Often, however, clarity can be enhanced by rewriting the condition without parentheses. For example, in the statement:

```
IF a = b AND (c = d OR e = f)
```

explicit grouping may be achieved by coding:

```
IF a = b AND c = d OR a = b AND e = f
```

Abbreviated Combined Conditions

Abbreviated combined conditions are conditions with implied subjects or implied operators. That is, the subject of the relation condition, or both the subject and the relational operator, may be omitted if they are the same as those in the preceding clause.

The format for an abbreviated combined condition is:

[NOT] relation-condition { { AND / OR } [NOT] [relation-operator] operand } ...

Either form of abbreviation may be used: the omission of subject, or the omission of subject and relational-operator. The effect of such abbreviations is that of inserting the previously stated subject in place of the omitted subject, or the previously stated relational operator in place of the omitted operator. All insertions terminate once a complete simple condition is encountered within a complex condition.

In all instances, the results must comply with the rules outlined in Table 4-4 above.

If the word NOT is used in an abbreviated condition, it is evaluated as follows:

- NOT participates as part of the relational operator if the word immediately following NOT is GREATER, >, LESS, <, EQUAL, or =.
- Otherwise NOT is interpreted as a logical operator with the result that the implied insertion of subject or relational operator results in a negated relation condition.

Below are examples of abbreviated combined conditions:

Abbreviated Combined and Negated Combined Relation Conditions	Expanded Equivalent
a = b OR c OR d	a = b OR a = c OR a = d
a > b AND NOT < c OR d	((a > b) AND (a NOT< c)) OR (a NOT <d)
NOT a = b OR c	(NOT (a = b)) OR (a = c)
a NOT EQUAL b OR c	(a NOT EQUAL b) OR (a NOT EQUAL c)
NOT (a GREATER b OR < c)	NOT ((a GREATER b) OR (a < c))
NOT (a NOT > b AND c AND NOT d)	NOT ((((a NOT > b) AND (a NOT > c)) AND (NOT (a NOT > d))))

Condition Evaluation Rules

The following order of logical evaluation is used to determine the truth value of a condition.

1. Conditions within parentheses are evaluated first. Within nested parentheses, evaluation proceeds from the least inclusive (innermost) condition to the most inclusive (outermost) condition.

2. Truth values for simple conditions are evaluated in the following order:

   Relation (following the expansion of any abbreviated relation condition)

   Class

   Condition-name

   Switch-status

   Sign

3. Truth values for negated simple conditions are established.

4. Truth values for combined conditions are established with this hierarchy:

   AND logical operators, followed by

   OR logical operators

5. Truth values for negated combined conditions are established.

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

The following examples illustrate the condition evaluation rules:

1. The condition below contains both AND and OR connectors.

```
IF X = Y AND FLAG = "Z" OR SWITCH = 0, GO TO PROCESSING.
```

Execution will be as follows, depending on various data values:

X	DATA Y	Value Flag	Switch	Executes
10	10	'Z'	1	YES
10	11	'Z'	1	NO
10	11	'Z'	0	YES
10	10	'p'	1	NO
6	3	'p'	0	YES
6	6	'p'	1	NO

2. A < B OR C = D OR E NOT > F: The evaluation is equivalent to (A < B) or (C = D) or NOT (E < F) and is true if any of the three individual parenthesized simple conditions is true.

3. The following time-card record description includes three condition-names. It is followed by an evaluation.

```
01 TIME-CARD
    05 EMP-STATUS PIC X.
      88 W VALUE 'W'.
      88 H VALUE 'H'.
      88 E VALUE 'E'.
    05 HOURS PIC 99.
     .
     .
     .

    IF W AND HOURS NOT = 0 ...
```

The evaluation is equivalent to:

```
IF (EMP-STATUS = 'W') AND NOT (HOURS = 0)...
```

and is true only if both the simple conditions are true.

4. A = 1 AND B = 2 AND G > -3 OR P NOT EQUAL TO "SPAIN"
is evaluated as:

```
[ (A = 1) AND (B = 2) AND (G > -3) ] OR NOT (P = "SPAIN")
```

If P = "SPAIN", the complex condition can be true only if all three of the following are true:

```
A = 1
B = 2
G > -3
```

However, if P is not equal to "SPAIN", the complex condition is true regardless of the values of A, B, and G.

5 The IDENTIFICATION DIVISION

IDENTIFICATION DIVISION

Function

The IDENTIFICATION division must be included in every COBOL program. It must appear as the first division in the source program. This division identifies the program. Additional user information, such as the date the program was written or the program author, may be included in the appropriate paragraph in the general format shown below.

Prime extension: this division may be preceded by asterisked comment lines.

Format

```
{ ID DIVISION.          }
{ IDENTIFICATION DIVISION. }

PROGRAM-ID. program-name. (no special characters in name)

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

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

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

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

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

[REMARKS. [comment-entry] ... ]
```

Syntax Rules

1. The IDENTIFICATION division must begin with the reserved words IDENTIFICATION DIVISION or ID DIVISION followed by a period and a space.

   Prime extension: ID may be used instead of IDENTIFICATION.

2. The PROGRAM-ID paragraph is required and must immediately follow the division header. The program-name is the name of the entry-point or object module, and is the name by which this program is referenced in a CALL statement. If it is omitted, MAIN is used by the compiler as the program-name.

3. The program-name follows the general rules for word formation in Chapter 4. It may be any alphanumeric string or data-name.

   Prime restriction: only the first eight characters of program-name are retained by the compiler. Because they define the entry-point name, these characters must be unique in any one runfile.

4. All remaining paragraphs are optional. A paragraph-header (a reserved word) identifies the type of information contained in each paragraph.

5. A comment-entry can be any combination of Prime characters. The continuation of a comment-entry by a hyphen in column 7 is not permitted; however, the comment-entry can appear on one or more lines. Any comment lines after the header line must be limited to Area B.

6. The REMARKS paragraph is a Prime extension.

7. Prime extension: optional paragraphs may appear in any order.

8. DATE-COMPILED causes the date and time of compilation to be printed in the listing file on the same line.

9. PROGRAM-ID and DATE-COMPILED may be used in the PROCEDURE division. In all references to PROGRAM-ID in the PROCEDURE division, the program name is substituted. In all references to DATE-COMPILED in the PROCEDURE division, the compilation date and time are substituted. Both substituted items are treated as nonnumeric literals.

   Examples are:

```
DISPLAY PROGRAM-ID.

MOVE DATE-COMPILED TO PIC-X-15.
```

EXAMPLE

This example forms one program with the examples following Chapters 6, 7, and 8. It shows the listing output for the ID division.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.      DISBURSE.
 AUTHOR.          ANNE LADD.
 INSTALLATION.    PRIME.
 DATE-WRITTEN.    SEPTEMBER 20, 1978.
 DATE-COMPILED.   820517 AT: 09:22
*****************************************************************
 REMARKS.  THIS PROGRAM PRODUCES A MONTHLY CASH DISBURSEMENTS
     JOURNAL:  A PRINTED DETAIL LIST AND TOTALS BY DEPARTMENTS
     WITH GRAND-TOTAL (CROSS-TOTAL) BALANCED AGAINST A JOB TOTAL.
*
     USE DETAIL-LINES WITH
         COL. 1-3  CHECK NO.,
         COL. 4-9  MMDDYY,
         COL. 13-32 VENDOR,
         COL. 33-35 DEPT. OR ACCT. NO.,
         COL. 36-42 AMOUNT.
     TO WRITE TAPE RECORD, ENTER YES  FOR TAPE REQUEST.
*
     THE PROGRAM CHECKS FOR INPUT ERRORS OF INVALID ACCOUNT
     NUMBER, INVALID DATE, INVALID NUMERIC FIELDS.  IT DOES NOT
     CHECK FOR SEQUENCE ERRORS IN DATE OR CHECK NUMBER, OR FOR
     DUPLICATE ENTRIES.  THE PROGRAM ASSUMES AN UNSORTED DATA FILE.
*****************************************************************
```

6
The ENVIRONMENT DIVISION

ENVIRONMENT DIVISION

Function

The ENVIRONMENT division defines those aspects of a program that are dependent upon hardware considerations.

Format

```
ENVIRONMENT DIVISION.
[ CONFIGURATION SECTION.
  [SOURCE-COMPUTER. [computer-name.]]
  [ OBJECT-COMPUTER. [computer-name] ]
  [ [object-computer-entry]. ]
  [ SPECIAL-NAMES. ]
  [ [special-names-entry]... ] ]
```

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

Syntax Rules

1. The ENVIRONMENT division must begin with the header ENVIRONMENT DIVISION, followed by a period and a space.

2. The mandatory sequence of required and optional paragraphs is shown in the above format.

   Prime extension: the clauses in the I-O-CONTROL paragraph may appear in any order.

3. Prime extension: the SOURCE-COMPUTER and OBJECT-COMPUTER entries and computer-name are optional.

4. Prime extension: the CONFIGURATION section is optional.

General Rules

1. Each section within the ENVIRONMENT division begins with its section-name, followed by the word SECTION, and each paragraph within each section begins with its paragraph-header.

2. This section may be used to document hardware-dependent features of a program.

3. The computer-name serves only as documentation. It is used to identify the computer upon which the COBOL program is to be compiled.

OBJECT-COMPUTER

Format

OBJECT-COMPUTER. [computer-name]

[, MEMORY SIZE integer { WORDS / CHARACTERS / MODULES }]

[, PROGRAM COLLATING SEQUENCE IS alphabet-name-1]

[, SEGMENT-LIMIT IS segment-number].

General Rules

1. The computer-name serves only as documentation. It is used to identify the computer on which the COBOL program will be executed.
2. The MEMORY SIZE clause serves as documentation only.
3. The SEGMENT-LIMIT clause serves as documentation only. The segment-number must be an integer ranging in value from 1 through 49.

SPECIAL-NAMES

This paragraph is required only if one or more of its statements is used.

Format

```
SPECIAL-NAMES.

  [CONSOLE IS mnemonic-name] ...

  [ switch-name [IS mnemonic-name]

    { 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-name-1 IS { STANDARD-1 }
                         { NATIVE     }
                         { EBCDIC     }  ] ...

  [ , CURRENCY SIGN IS literal]

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

General Rules

1. The mnemonic-name is a programmer-defined word that will be associated with CONSOLE throughout the program. The following example uses TTY as the mnemonic-name. The coding would cause the field YEAR OF HIRE-DATE to be displayed on the console.

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
          .
          .
          .
SPECIAL-NAMES.  CONSOLE IS TTY.
          .
          .
          .
PROCEDURE DIVISION.
          .
          .
          .
    DISPLAY YEAR OF HIRE-DATE UPON TTY.
```

2. There are eight switch-names, which are reserved words in Prime COBOL 74:

```
CBLSW0
CBLSW1
CBLSW2
CBLSW3
CBLSW4
CBLSW5
CBLSW6
CBLSW7
```

In a COBOL 74 program, switches may be tested but not set. Switches may be set only in response to requests issued at runtime. (See Chapter 3.) Their purpose is to allow for changes in the real world each time a program is run. As an example, a switch can determine whether or not to add month-end processing. Switch condition-names or switch-status-conditions are programmer-defined names. At least one condition-name must be associated with each switch used in a program, so that either the ON or the OFF status name is required. (See Chapter 4.) The status of a switch is determined by testing an associated condition-name, as in the following example.

Switch-names may be qualified by mnemonic-names (status-names), as SWITCH-ON OF SWITCH-ONE in this example.

```
SPECIAL-NAMES.
    CBLSW0 IS TAPE-SWITCH,
              ON STATUS IS TAPE-SWITCH-ON,
              OFF STATUS IS TAPE-SWITCH-OFF,
    CBLSW1 IS SWITCH-ONE,
              ON STATUS IS SWITCH-ON,
              OFF STATUS IS SWITCH-OFF,
    CBLSW2 IS SWITCH-TWO,
              ON STATUS IS SWITCH-ON,
              OFF STATUS IS SWITCH-OFF,
    .
    .
    .
TEST SECTION.
  ONLY-PARAGRAPH.
    IF TAPE-SWITCH-OFF DISPLAY 'TAPE CANNOT BE PROCESSED'.
    IF SWITCH-ON OF SWITCH-ONE DISPLAY 'NO PRINT-OUT'.
```

3. The alphabet-name clause is syntax-checked only.

4. The literal represents the currency sign to be used in the PICTURE clause. It is a single-character, nonnumeric literal that will be used to replace the dollar sign as the currency sign. The designated character may not be a single or double quote mark, or any of the characters defined for PICTURE representations.

5. The clause DECIMAL-POINT IS COMMA means that the functions of comma and period are exchanged in the character-string of PICTURE clauses and in numeric literals.

INPUT-OUTPUT SECTION

The INPUT-OUTPUT SECTION is used when there are external data files. It allows specification of peripheral devices and information needed to transmit and handle data between the devices and the program. The section has two paragraphs: FILE-CONTROL and I-O-CONTROL.

FILE-CONTROL

Each file requires one file-control-entry. The format chosen is dependent upon file organization.

Format 1

```
SELECT [OPTIONAL] file-name

  ASSIGN TO device-name

[ ; RESERVE integer-1 [ AREA  ] ]
                      [ AREAS ]

[ ; ORGANIZATION IS SEQUENTIAL]

[ ; ACCESS MODE IS SEQUENTIAL]

[ ; FILE STATUS IS data-name-1].
```

Format 2

```
SELECT file-name

  ASSIGN TO PFMS

[ ; RESERVE integer-1 [ AREA  ] ]
                      [ AREAS ]

  ; ORGANIZATION IS RELATIVE

[ ; ACCESS MODE IS { SEQUENTIAL [, RELATIVE KEY IS data-name-1]                    }
                   { { RANDOM  } , RELATIVE KEY IS data-name-1                     }
                   { { DYNAMIC }                                                   } ]

[ ; FILE STATUS IS data-name-2].
```

Format 3

```
SELECT file-name
  ASSIGN TO PFMS
[ ; RESERVE integer-1 [ AREA ] ]
                      [ AREAS ]
  ; ORGANIZATION IS INDEXED
[ ; ACCESS MODE IS { SEQUENTIAL } ]
                   { RANDOM     }
                   { DYNAMIC    }
  ; RECORD KEY IS data-name-1
[ ; ALTERNATE RECORD KEY IS data-name-2 [WITH DUPLICATES]] ...
[ ; FILE STATUS IS data-name-3].
```

General Rules

1. The SELECT clause must be specified first in the file-control entry. The following clauses may appear in any order.

2. The file-name is a programmer-defined name described in the DATA division. Each file specified here must have a file description entry in the DATA division. The ASSIGN clause associates the file-name with a storage medium or input/output hardware. The device-names are Prime COBOL 74 reserved words. Allowable device-names appear in Table 6-1.

Table 6-1
Device Specifications

Device-name	Hardware Device
TERMINAL	CRT terminal TTY terminal
READER	Card reader (for future designation)
PRINTER	System printer (goes to disk, can be spooled)
PUNCH	Card punch (for future designation)
MT7 and MT9	9-Track magnetic tape drive
PFMS	Disk storage (Prime File Management System)
OFFLINE-PRINT	Serial line printer

Examples:

```
SELECT SCREEN-FILE ASSIGN TO TERMINAL.
SELECT DISK-FILE ASSIGN TO PFMS.
SELECT TAPE-FILE1 ASSIGN TO MT9.
```

3. The OPTIONAL clause may be specified only for input files. It is syntax-checked only.

4. The ASSIGN clause specifies the association of the file referred to by file-name with a storage medium.

5. The RESERVE clause is for documentation only. Whether or not it is used, one buffer area will be assigned by the compiler.

6. The ORGANIZATION clause specifies the logical structure of a file. When the clause is omitted, the default is SEQUENTIAL.

7. The sequence in which records are accessed is described through the use of the ACCESS MODE clause. When this clause is omitted, the default is SEQUENTIAL.

8. The RECORD KEY clause is discussed in Chapters 12 and 13, INDEXED SEQUENTIAL FILES and RELATIVE FILES.

9. The ALTERNATE RECORD KEY clause is discussed in Chapter 12, INDEXED SEQUENTIAL FILES.

10. The FILE STATUS clause permits the user to specify a two-character field (data-name-3), described in the WORKING-STORAGE section or the LINKAGE section, as the file status field. Data-name-3 may be qualified.

    Prime extension: data-name-3 may be either alphanumeric or an unsigned numeric display field.

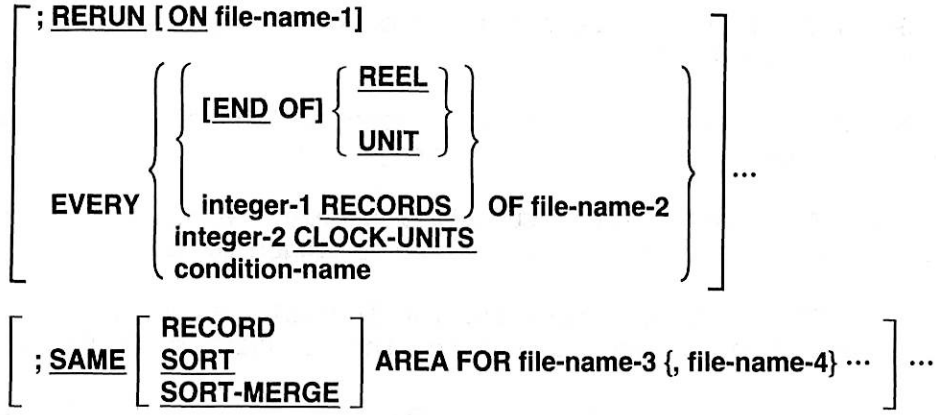
    When the FILE STATUS clause is specified in the FILE-CONTROL paragraph, COBOL file control moves a value into data-name-3 after the execution of every statement that refers to that file. Thus, the FILE STATUS data item is updated during the execution of the OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, or START statements. The value in data-name-3 indicates to the COBOL program the status of execution of the statement.

    Valid combinations of file status values for each type of file organization are shown in Table A-5 of Appendix A.

I-O-CONTROL

Format

I-O-CONTROL.



Syntax Rule

The I-O-CONTROL paragraph is optional.

General Rules

1. SAME AREA is treated as SAME RECORD AREA. The SAME AREA or SAME RECORD AREA clause allows the programmer to share the same memory areas for files that are not sort or merge files. This feature saves memory space and eliminates MOVEs from one record to another, thus saving execution time. No file may be listed in more than one SAME AREA or SAME RECORD AREA clause.

   An example is given in the sample program in Chapter 12.

2. The SAME AREA or SAME RECORD AREA clause specifies that two or more files are to use the same memory area for processing of the current logical record. All of the files may be open at the same time. A logical record in the SAME RECORD AREA is considered both as a logical record of each opened output file whose file-name appears in this SAME RECORD AREA clause, and as a record of the most recently read input file whose file-name appears in this SAME RECORD AREA clause. This is equivalent to an implicit redefinition of the area, that is, records are aligned on the leftmost character position.

3. If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in the first clause must appear in the second clause. However, additional file-names not appearing in that SAME AREA clause may also appear in that SAME RECORD AREA clause.

4. The files referenced in the SAME AREA or SAME RECORD AREA clause need not all have the same organization or access.

5. The RERUN clause is checked for syntax only.

EXAMPLE

This sample forms one program with the samples at the end of Chapters 5, 7, and 8.

```
    ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
    SOURCE-COMPUTER.  PRIME.
    OBJECT-COMPUTER. PRIME,
    INPUT-OUTPUT SECTION.
    FILE-CONTROL.
        SELECT TAPE-FILE, ASSIGN TO MT9.
        SELECT PRINT-FILE, ASSIGN TO PRINTER.
        SELECT DISK-FILE, ASSIGN TO PFMS.
    ****************************************************************
```

7
The DATA DIVISION

DATA DIVISION

Function

The DATA division of the COBOL source program defines the nature and characteristics of the data to be processed by the program. Data to be processed falls into three categories:

- Data in files, which enters or leaves the internal memory of the computer from or to a specified storage area or areas
- Data developed internally and placed into intermediate or working storage
- Data passed to the program from a calling program

The DATA division consists of three optional sections. If used, they must appear in the following order:

1. FILE section. Files and records in files are described.
2. WORKING-STORAGE section. Memory space is defined for the storage of items that are not part of external data files but are intermediate processing results.
3. LINKAGE section. Data available to both a called program and a calling program is described in the called program.

Format

```
DATA DIVISION.

[ FILE SECTION.
  [ file-description-entry.
    [record-description-entry] ... ] ...
  [ sort-merge-file-description-entry.
    {record-description-entry} ... ] ... ]

[ WORKING-STORAGE SECTION.
  [ level-77-data-description-entry
    data-description-entry ] ... ]

[ LINKAGE SECTION.
  [ level-77-data-description-entry
    data-description-entry ] ... ]
```

Syntax Rules

1. The DATA division must begin with the header DATA DIVISION, followed by a period and a space.

2. When included, optional sections of the DATA division must be in the order shown above.

3. Prime extensions: file-description-entries and sort-merge file-description-entries may appear in any order. In WORKING-STORAGE, level-01 and level-77 items may be described in any order.

General Rules

1. Each section within the DATA division begins with its section-name, followed by a period and a space.

2. In WORKING-STORAGE, a data-description-entry uses the same format as a record-description-entry.

FILE SECTION

Function

The FILE section of the DATA division defines the structure of data files. Each file is defined by a file-description-entry (FD) or a sort-merge-file-description-entry (SD), and by one or more associated record-description-entries.

Format

FILE SECTION.

```
[ file-description-entry. [record-description-entry] ...                ]
[                                                                       ] ...
[ sort-merge-file-description-entry. {record-description-entry} ...     ]
```

Syntax Rules

1. The FILE section is optional. If used, it must begin with the header FILE SECTION, followed by a period and a space.

2. The FILE section contains FD and SD entries. Each one should be followed immediately by one or more associated record-description-entries. There is no limit to the number of FD and SD entries in the FILE section. The number of files that can be opened at once is limited only by PRIMOS. This limitation is given in Appendix J.

General Rule

Each FD or SD entry must be associated with an I-O device by a SELECT statement in the ENVIRONMENT division.

Note

The formats and the clauses required in an FD entry for a nonsort file are described in this chapter. For a complete discussion of an SD entry for a sort-merge file, see Chapter 11, THE SORT-MERGE MODULE.

FILE-DESCRIPTION-ENTRY

Function

The file description provides information concerning the physical structure, identification, and record names of a nonsort file.

Format

```
FD file-name [ COMPRESSED ]
             [ UNCOMPRESSED ]

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

[ ; CODE-SET IS alphabet-name]

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

[ ; LABEL { RECORD IS  } { STANDARD } ]
          { RECORDS ARE } { OMITTED  }

[ ; OWNER IS literal-1]

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

[ ; VALUE OF { { FILE-ID  } IS { data-name-3 } } ... ]
             { { OWNER-ID }    { literal-2   } }
             { { VOL-ID   }                    }

    [record-description-entry] ...
```

Syntax Rules

1. The level indicator FD identifies the beginning of a file description and must precede the file-name.

2. The file-name follows the general rules in WORD FORMATION in Chapter 4.

3. The COMPRESSED/UNCOMPRESSED option is used only with sequential input files and is discussed below. For -OLD, the default is COMPRESSED; for normal I-O, the default is UNCOMPRESSED.

4. The file-description-entry is a sequence of clauses that must be terminated by a period. There is no restriction on the number of file-description-entries; the number of files that may be opened at one time is limited by PRIMOS. This limitation is listed in Appendix J.

5. If the DATA RECORD clause is used, one or more record-description-entries must follow the file-description-entry. Record-description-entries are presented with a detailed format on page 7-16.

6. All clauses that follow file-name are optional.

COMPRESSED/UNCOMPRESSED — PRIME EXTENSION

Function

The UNCOMPRESSED option enables a disk READ or WRITE based on record length, while the COMPRESSED option enables a READ or WRITE by compression control characters.

Format

FD file-name [**COMPRESSED** | **UNCOMPRESSED**]

General Rules

1. If no option is specified, the default is UNCOMPRESSED. For COBOL programs written for Rev. 18 and earlier software revisions, use the -OLD switch in the compiling command line so that the default will be COMPRESSED, or add COMPRESSED to the FD line.

2. The UNCOMPRESSED option must be used when reading sequential I-O files containing nondisplay numeric data, such as packed or binary data.

Note

Compression is the elimination of multiple blank characters. File compression is effected by replacing any string of three or more blank characters with a control character plus a count. When a record is written with compression control, the first space character in such a string is replaced by the ASCII control character DC1, and the second space is replaced by a binary count (3 through 255) of spaces in the string. The 3rd through 255th spaces are then deleted. When the same record is read with compression control, each combination of DC1 plus number is replaced by that number of spaces before the record is made available to the program.

The line-feed character (LF), rather than a word count, is used to define the end of each record in a file read or written with compression control.

BLOCK CONTAINS

Function

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

Format

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

Syntax Rules

1. The BLOCK CONTAINS clause is optional.

2. The clause can be used only in connection with tape files. PRIMOS disk files do not require explicit blocking for efficient access.

Note

Chapter 14 discusses the BLOCK CONTAINS clause in more detail.

CODE-SET

Function

The CODE-SET clause specifies the character code set used to represent data on the external media.

Format

CODE-SET IS alphabet-name

DATA RECORDS

Function

The DATA RECORDS clause serves only as documentation for the names of data records with their associated file.

Format

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

Syntax Rule

The data-names 1 and 2 are the names of the data records for the FD entry. They must be specified by 01-level items following the file description and must follow the general rules for word formation in Chapter 4.

General Rules

1. The presence of more than one data-name indicates that the file contains more than one type of data record. These records may have different sizes and formats. The order in which they are listed is not significant.

2. Conceptually, all data records within a file share the same area, regardless of the number or types of data records within the file.

LABEL RECORDS

Function

The LABEL RECORDS clause specifies whether labels are present for the file.

Format

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

General Rules

1. If this clause is not present, LABEL RECORDS OMITTED is assumed.
2. OMITTED specifies that no explicit labels exist for the file or device to which the file is assigned.
3. STANDARD specifies that a label exists for the file and that the label conforms to system specifications.
4. Each Prime device requires a specific LABEL option, as shown in Table 7-1 below.

Note

See the Magnetic Tape User's Guide and Chapter 14 of this manual on writing standard labels for magnetic tape.

Table 7-1
LABEL Clause Requirement

Device	Standard	Omitted
TERMINAL		X
READER		X
PRINTER		X
PUNCH		X
MT7 or MT9 (TAPE)	X	X
PFMS (DISK)		X

OWNER IS -- PRIME EXTENSION

Function

The OWNER IS clause specifies the pathname of the user file directory (UFD) in which the file is contained.

Format

OWNER is literal-1

Syntax Rules

The OWNER IS clause may be used only with disk files, and only in conjunction with a VALUE OF FILE-ID clause.

General Rules

1. The literal-1 is a nonnumeric value. With -OLD, it must not exceed six characters. For normal I-O, VALUE OF FILE-ID IS data-name should be used instead of OWNER IS to give a long pathname.
2. The clause may be overridden by file assignments at runtime if the -OLD compile option was used. See Appendix K.
3. If the clause is used, it must follow the above rules. If it is omitted, the current UFD is used as default.
4. The OWNER IS clause must not be used if the VALUE OF FILE-ID clause is followed by a data-name instead of a literal.
5. If both FILE-ID IS literal-2 and OWNER IS literal-1 are used, the pathname sought is literal-1>literal-2.

Note

This feature is included only for compatibility with older versions of Prime COBOL, and should not be used with normal I-O.

RECORD CONTAINS

Function

The RECORD CONTAINS clause specifies the size of data records.

Format

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

General Rules

1. This clause is always optional, since the size of each data record is defined fully by the set of data-description-entries constituting the record (level 01) declaration.

2. The integer-4 may not be used by itself unless all the data records in the file have the same size. In this case, integer-4 represents the exact number of characters in the data record. If integer-3 and integer-4 are both used, they refer to the minimum number of characters in the smallest size data record, and the maximum number of characters in the largest size data record, respectively.

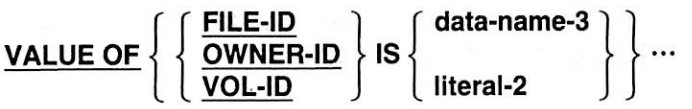
3. The maximum size of a single data record is listed in Appendix J.

VALUE OF FILE-ID, VOL-ID, OWNER-ID

Function

The VALUE OF clause associates the internal file-name with a PRIMOS file, thus allowing for the linkage of internal and external file-names.

Format



Syntax Rules

1. Disk file-names in data-name-3 and literal-2 must have the following format:

   [[[MFD-name>] UFD-name>] sub-UFD-name>...] file-name

   where the file-name is the PRIMOS file name.

2. Tape file-names in data-name-3 and literal-2 must have the following format:

   drivename, label-type, owner-id, volume-id

drivename	$MT(x), where x is a drive number from 0 through 3 (0 through 7 if logical drives were assigned).
label-type	N: no label information. S: standard labels.
owner-id	A 14-character field. This is called the tape file-id by LABEL. This is not the same as the OWNER field of the LABEL command.
volume-id	A six-character field that is written in the label of the tape being created, or is checked if the tape is being read. This is also called the volume serial number (VSN).

   Chapter 14 discusses tape files more thoroughly.

General Rules

Note

The section on OWNER IS earlier in this chapter includes some restrictions on VALUE OF FILE-ID when used in conjunction with OWNER IS.

1. The literal is a nonnumeric literal. For compatibility with other COBOLs, the FILE-ID clause may be used in conjunction with the OWNER IS clause presented earlier.

2. The data-name must be in the WORKING-STORAGE section. It may be qualified, but it must not be subscripted, indexed, or described with USAGE IS INDEX. The value of the data-name must not exceed 120 characters for FILE-ID.

3. If there is no VALUE OF clause, the compiler will use the file-name following FD.

4. If there is a VALUE OF clause, the compiler will use the literal or the value in the data-name as the name of the PRIMOS file or pathname. If the data-name is used, the pathname should be contained in the data-name. The COBOL program must assign a value to the data-name.

5. The data-name-3 may be assigned a value with ACCEPT statements. For an example, see the sample program at the end of Chapter 13.

6. VOL-ID and OWNER-ID are reserved for future implementation.

7. For rules with -OLD, see Appendix K.

8. If the value of the data-name is changed, the file must be closed and then reopened in order for the new data-name to be in effect.

Examples

A PFMS file named FILEX can be associated with a logical COBOL file named TEST-FILE in any of the following ways.

1. Value is literal:

```
FD TEST-FILE
   LABEL RECORDS STANDARD
   VALUE OF FILE-ID 'FILEX'.
```

2. Value is data-name:

```
FD TEST-FILE
   LABEL RECORDS STANDARD
   VALUE OF FILE-ID IS TFILE-NAME.
    .
    .
WORKING-STORAGE SECTION.
77 TFILE-NAME                PIC X(24).
```

An actual file-name can be associated with the logical file-name TEST-FILE by executing COBOL statements. For example:

```
IF NEW-FILE = 1
            MOVE "FILEX" TO TFILE-NAME,
ELSE IF NEW-FILE = 2
            MOVE "OTHER" TO TFILE-NAME,
     ELSE
            MOVE "ARTHUR>TESTFILE PASSWORD" TO TFILE-NAME.
```

Another way to do it could be:

```
MOVE SPACES TO TFILE-NAME
DISPLAY "ENTER TEST-FILE NAME."
ACCEPT TFILE-NAME.
```

RECORD-DESCRIPTION-ENTRY

Format 1

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

 [ ; BLANK WHEN ZERO]

 [ ; { JUSTIFIED } RIGHT ]
     { JUST      }

 [ ; OCCURS { integer-1 TO integer-2 TIMES DEPENDING ON data-name-2 }
            { integer-2 TIMES                                       }

   [ { ASCENDING  } KEY IS data-name-3 [, data-name-4] ... ] ...
   { DESCENDING }

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

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

 [ ; REDEFINES data-name-5]

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

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

 [                 COMPUTATIONAL
                   COMP
                   COMPUTATIONAL-1
                   COMP-1
   ; [USAGE IS]    COMPUTATIONAL-2
                   COMP-2
                   COMPUTATIONAL-3
                   COMP-3
                   DISPLAY
                   INDEX                ]

 [ ; VALUE IS literal]
```

Format 2

```
66 data-name-1 ; RENAMES data-name-2 [ { THROUGH } data-name-3 ]
                                      { THRU    }
```

Format 3

```
88 condition-name; VALUE IS literal-1 [ { THROUGH } literal-2 ]
                                      { THRU    }

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

Syntax Rules

1. The level-number in Format 1 may contain a value of 01 through 49, or 77.

2. In Format 1, clauses can be written in any order with two exceptions: the data-name-1 or FILLER clause must immediately follow the level-number; and the REDEFINES clause, when used, must immediately follow the data-name-1 clause.

3. In Format 1, the PICTURE clause must be specified for every elementary item except when USAGE is described as binary (COMPUTATIONAL) or floating-point (COMPUTATIONAL-1 or COMPUTATIONAL-2) or INDEX. A group item cannot contain a PICTURE clause.

4. The OCCURS clause cannot be specified in a data-description-entry that has a 66, 77, or an 88 level-number.

5. Format 2 permits alternative, possibly overlapping groups of elementary items.

6. The words THRU and THROUGH are equivalent.

General Rules

A record-description-entry can appear in the FILE, WORKING-STORAGE, or LINKAGE section of the DATA division. All records in each file-description-entry must be described by record-description-entries.

LEVEL-NUMBER

Function

The level-number shows the position of a data-item within the hierarchy of data in a logical record. It also specifies entries for condition-names, the RENAMES clause, and data items in the WORKING-STORAGE and LINKAGE sections.

Format

level-number

Syntax Rules

1. A level-number is required as the first element in each data-description-entry. (See RECORD-DESCRIPTION-ENTRY.)

2. Data-description-entries subordinate to an FD or SD entry must have level-numbers 66, 88 or 01 through 49.

3. Data-description-entries in the WORKING-STORAGE and LINKAGE sections must have level-numbers 66, 77, 88 or 01 through 49.

General Rules

1. Level-numbers are used to subdivide a record so that each item in the record may be referred to. A record can be divided, and each subdivision further divided. An item that is not further subdivided is called an elementary item. A record can itself be an elementary item.

2. A group consists of one or more consecutive group or elementary items; groups can, in turn, be combined into other groups. In standard COBOL, a group consists of a specified group item and all following items until the next item with a level-number less than or equal to that of the group item is reached.

   Prime extension: within a group item numbered higher than 01, sub-items need not be numbered in the same or successively higher levels. This numbering, however, does not conform to the ANSI standard. For example, the following two declarations are equivalent.

Standard Use

```
01 RECORD-1.
  05   C-1.
    10 AA      PIC 99.
    10 BB      PIC 9V9.
  05   C-2.
    10 CC      PIC 9(5).
    10 DD      PIC X(8).
```

Substandard Use

```
01 RECORD-1.
  5    C-1.
    10 AA      PIC 99.
     8 BB      PIC 9V9.
  03   C-2.
    20 CC      PIC 9(5).
    10 DD      PIC X(8).
```

3. The level-numbers range from 01, the most inclusive level, to 49, the least inclusive level. Any level-number except 49 can denote a group.

4. The level-number 01 identifies the first entry in each record description. A reference to a level-01 data-name in the PROCEDURE division is a reference to the entire record.

5. Multiple level-01 entries subordinate to one FD represent implicit redefinitions of the same area.

6. The following special level-numbers have been assigned to certain entries where there is no real concept of hierarchy.

   - The level-number 77 is assigned to identify noncontiguous WORKING-STORAGE or LINKAGE data items. They may be used only as described in Format 1 of the record-description-entry.

     Level-77 data items are elementary items that cannot be subdivided.

   - Level-number 88 is assigned to entries that define condition-names associated with a conditional variable. Condition-names and the conditional variable are discussed in Chapter 4. Level 88 can be used only with Format 3 of the record-description-entry.

     Level-88 entries can contain individual values, series of individual values, or a range of values.

     Example:

```
01  TEST-AREA PIC X.
    88  TEST-VALUE-1 VALUE '1'.
    88  TEST-VALUE-2 VALUE '1', '2'.
    88  TEST-VALUE-3 VALUE '1' THRU '8'.
    88  TEST-VALUE-4 VALUE '1' THRU '4', '6', '7'.
```

     The VALUE clause is required in a level-88 entry, and must be the only clause in the entry. THRU and THROUGH are equivalent.

A level-88 entry must be preceded either by another level-88 entry, or by an elementary item, called the conditional variable.

The condition-name may be qualified by the name of the conditional variable. A condition-name is used in the PROCEDURE division in place of a relational condition, as discussed in Chapter 4. A condition-name, when written in the PROCEDURE division, must be subscripted if its conditional variable is subscripted. The type of literal in a condition-name VALUE clause must be consistent with the data type of the conditional variable.

In the following example, PAYROLL-PERIOD is the conditional variable. The picture associated with it limits the value of the 88 condition-name to one digit.

```
02  PAYROLL-PERIOD      PICTURE IS 9.
    88  WEEKLY              VALUE IS 1.
    88  SEMI-MONTHLY        VALUE IS 2.
    88  MONTHLY             VALUE IS 3.
```

Using the above description, you may write the procedural condition-name test:

```
IF MONTHLY PERFORM DO-MONTHLY.
```

An equivalent statement is:

```
IF PAYROLL-PERIOD = 3 PERFORM DO-MONTHLY.
```

For an edited elementary item, values in a VALUE clause must be expressed in the form of nonnumeric literals.

- The level-number 66 is assigned to identify RENAMES entries, which are discussed later in this chapter. It can be used only with Format 2 of the record-description-entry.

Example

The weekly time card record in Figure 7-1 illustrates the level concept. It is divided into four major items: name, employee-number, date, and hours, with more specific information appearing for name and date.

TIME-CARD	NAME	LAST-NAME FIRST-INIT MIDDLE-INIT
	EMPLOYEE-NUM	
	DATE-STARTED	MONTH DAYY YEAR
	HOURS-WORKED	

Weekly Time-card Record
Figure 7-1

The time-card record might be described by DATA division entries having the following level-numbers, data-names, and picture definitions.

```
01  TIME-CARD.
    05  NAME.
        10   LAST-NAME          PICTURE X(18).
        10   FIRST-INIT         PICTURE X.
        10   MIDDLE-INIT        PICTURE X.
    05  EMPLOYEE-NUM           PICTURE 99999.
    05  DATE-STARTED.
        10   MONTH              PIC 99.
        10   DAYY               PIC 99.
        10   YEAR               PIC 99.
    05  HOURS-WORKED           PICTURE 99V9.
```

DATA-NAME OR FILLER

Function

A data-name specifies the name of the data being described. FILLER specifies an elementary item that cannot be referred to explicitly.

Format

```
{ data-name }
{ FILLER    }
```

General Rules

1. Prime extension: FILLER may be used to name a group item as well as an elementary item.

2. A FILLER item cannot be referred to explicitly. However, FILLER can be used as a conditional variable because such use does not require explicit reference to the FILLER item, but rather to its value.

3. A VALUE clause can be used with a FILLER item.

Example

```
01  INPUT-RECORD.
    05  FILLER                          PIC 9.
        88 FULL-TIME   VALUE 1.
        88 PART-TIME   VALUE 2.
    05  NAME                            PIC X(40).
    05  FILLER                          PIC X(10).
    05  HOURS                           PIC 99.
```

In this example, the FILLER items cannot be changed except by moving data into their group item INPUT-RECORD. The second FILLER item cannot be referred to individually, but the first FILLER item can be tested by code such as the following:

```
IF FULL-TIME PERFORM 070-FULL-TIME.
```

BLANK WHEN ZERO

Function

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

Format

BLANK WHEN ZERO

Syntax Rule

The BLANK WHEN ZERO clause can be used only for an elementary numeric or numeric edited item.

General Rules

1. The BLANK WHEN ZERO clause specifies that the data item will be set to blanks when the value is all zeros. Leading zeros are not suppressed by this clause. Figure 7-2 illustrates some uses for this clause.

2. If the clause is specified for a numeric item, the category of the item is interpreted as numeric edited.

3. An asterisk used as the zero-suppression symbol may not be used in a picture-string with this clause.

Value	Description of OUT-COST		Result
0012.34	9999.99	BLANK WHEN ZERO	0012.34
0123.45	$9999.99	BLANK WHEN ZERO	$0123.45
01.2345	$9999.99	BLANK WHEN ZERO	$0001.23
0000.04	$$$$$.99	BLANK WHEN ZERO	$.04
0000.00	$$$$$.99	BLANK WHEN ZERO	bbbbbbbb
0000.00	$$$$$.99		$.00
0000.00	ZZZZVZZ	BLANK WHEN ZERO	bbbbbb
0000.04	ZZZZVZZ	BLANK WHEN ZERO	4
0000.00	ZZZZ.ZZ	BLANK WHEN ZERO	bbbbbbb
0000.04	ZZZZ.ZZ	BLANK WHEN ZERO	.04
0000.00	ZZZZ.99	BLANK WHEN ZERO	bbbbbbb
0000.00	ZZZZ.99		.00

Examples: BLANK WHEN ZERO
(b = blank)
Figure 7-2

JUSTIFIED

Function

The JUSTIFIED clause specifies right alignment of data within a field.

Format

```
{ JUSTIFIED } RIGHT
{ JUST      }
```

Syntax Rules

1. This clause can be specified only at the elementary level.
2. JUST is a valid abbreviation of JUSTIFIED.
3. The JUSTIFIED clause cannot be used for data items described as numeric, or for those for which editing is specified.

General Rules

1. When the JUSTIFIED clause is included, values are stored in right-to-left fashion. In a MOVE operation, if the sending field is shorter than the JUSTIFIED receiving field, space filling occurs in the leftmost positions of the receiving field. If the sending field is longer, the leftmost characters of the sending field are truncated. If the sending field is the same size as the JUSTIFIED receiving field, the result is a straight MOVE, including spaces.
2. When the JUSTIFIED clause is omitted, the standard alignment rules in Chapter 4 apply.

Examples

Of the following two fields, one is justified and one is not. Since the following MOVE statement, however, involves two fields of the same size, the contents of field Y are not right-justified. (b = blank.)

```
    01 Y                PIC X(4) JUST.
    01 Z                PIC X(4).

MOVE 'AB' TO Z. MOVE Z TO Y.
EXHIBIT Z ' ' Y.        terminal displays:    Z = ABbb   Y = ABbb
```

OCCURS

Functions

The OCCURS clause permits the definition of related sets of repeated data, such as tables, arrays, and lists. It supplies required information for the application of subscripts or indexes.

Format

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

[ { ASCENDING  } KEY IS data-name-3 [, data-name-4] ... ] ...
  { DESCENDING }

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

Note

See Chapter 10, TABLE HANDLING, for detailed discussion of the OCCURS clause.

PICTURE

Function

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

Format

```
{ PICTURE } IS character-string
{ PIC     }
```

Syntax Rules

1. A PICTURE clause can be specified only at the elementary item level.

2. A picture character-string or picture-string consists of certain allowable combinations of characters in the COBOL character set used as symbols. The allowable combinations determine the category of the elementary item.

3. The maximum number of characters allowed in a picture-string is 32. In the shorthand notation X(n), the repeat integer n may not exceed 32767, and may not be less than 1.

4. The PICTURE clause must be specified for every elementary item except binary and floating-point items. The clause is optional for items with the COMP clause. (The default is PIC S9999.) PICTURE is not allowed with items whose usage is COMPUTATIONAL-1, COMPUTATIONAL-2, or INDEX.

5. PIC is an abbreviation for PICTURE.

6. A picture-string must include at least one of the characters Z A * X 9 or at least two consecutive appearances of the characters + - $.

General Rules

Categories: Five categories of data can be defined with a PICTURE clause: alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited. Their picture-strings have the following restrictions.

- Alphabetic: The picture-string can contain only the characters A and B. Contents of the data field so described may only be any combination of the letters of the English alphabet and the COBOL space character. See Example 3 of Figure 7-3.

Example	Value	Nonedited Picture	Stored as
1	37	P999	0
	.5	P999	0
	.05	P999	.05
2	37	999PP	0
	3700	999PP	3700

Example	Value	Edited Picture	Will Print as
3	JDOE	ABAAAA	J DOE
4	JDOE	XBXXXX	J DOE
5	113113	XXX0XXX	1130113
6	459243333	XXX/XX/XXXX	459/24/3333
7	10000	99,999	10,000
8	9999	999.9	999.0
9	999	999.9	999.0
10	+5500	+9999	+5500
11	-5500	+9999	-5500
12	+5500	9999CR	5500
13	+5500	9999DB	5500
14	+5500	9999+	5500+
15	+5500	ZZZ.99	500.00
16	123	$$$.$$	$23.00
17	123	$$$.99	$23.00
18	003	$$$.$$	$3.00
19	000	$$$.$$	
20	00345	$$$.$$	$45.00
21	0000	ZZ,ZZZ	
22	00345	ZZ,ZZZ	345
23	0000	$*****	$*****
24	0000	$ZZZZZ	$
25	00	$***.99	$***.00
26	00	ZZ,ZZZ.99	.00
27	0.03	ZZ,ZZZ.ZZ	.03
28	0.00	ZZ,ZZZ.ZZ	

Examples of PICTURE Clauses and Conversions
Figure 7-3

- Numeric: The picture-string can only contain the symbols 9, P, S, and V. The number of digit positions that may be represented by this picture-string is 1 to 18; the contents of this field may only be a combination of the digits 0 through 9, plus an optional sign.

- Alphanumeric: The picture-string is a combination of the data description characters X, A, or 9, and the item is treated as if the string contained all X's. Alphanumeric picture-strings may not employ all 9's or all A's. Item contents may be any character from the computer's ASCII character set.

- Alphanumeric edited: The picture-string is restricted to certain combinations of the following symbols: A, X, 9, B, 0, /. Allowable combinations are discussed in Symbols below. Contents of the field may be any character from the computer's ASCII character set.

- Numeric edited: The picture-string is a certain combination of the editing symbols Z . CR DB , $ + * B 0 - / 9 V P. It must contain at least one of the editing symbols Z . CR DB , + * B 0 - / and field contents must be numerals. The maximum number of digit positions is 18.

Size: The size of an elementary item (the number of character positions occupied by the item in standard data format) is determined by the number of symbols that represent character positions, as listed below.

An integer enclosed in parentheses, following the symbols A X 9 P Z * B / 0 + - or the currency symbol, indicates the number of consecutive occurrences of that symbol. This is the repeat integer.

Symbols: Symbols used in a picture-string to define an elementary item have the following functions:

A Each A represents a character position containing only a letter of the alphabet, or a space.

B Each B represents a character position into which a space character will be inserted. Its use is explained in Editing Rule 3 below.

P Each P indicates an assumed decimal scaling position. It specifies the location of an assumed decimal point that does not appear in the data item. The P is not counted in the size of the data item, but is counted in determining the maximum number of digit positions in numeric-edited items or numeric items.

In conversions, each digit position described by a P is considered to contain the value zero. The assumed decimal point is considered to be to the left of the P that is leftmost in a string, or to the right of P's that are rightmost in a string. Thus the effect of each P is to divide or multiply a data item by one power of ten.

The scaling position character P may appear only to the left or right of the other characters in the string, except that the sign character S and the assumed decimal point V may appear to the left of a leftmost string of P's. Since the character P implies an assumed decimal point, the symbol V is redundant as either the leftmost or rightmost character within such a PICTURE description.

The character P and the insertion character period (.) may not both occur in the same picture-string.

See the two nonedited examples in Figure 7-3.

S The picture-string symbol S indicates the presence of a sign in a data item, but implies nothing about the actual format or location of the sign in storage.

The symbol S is not counted in determining the size of the elementary item, unless the entry is subject to a SIGN clause with the SEPARATE qualifier. (See SIGN.)

When used, the S symbol must be written as the leftmost character in a picture-string.

V The character V indicates the position of an assumed decimal point. Since a numeric item cannot contain an actual decimal point, an assumed decimal point is used to provide information concerning the alignment of items involved in computations. The V does not represent a character position and therefore is not counted in the size of the item. Only one V is permitted in any single picture.

X Each X represents a character position that may contain any allowable character from the computer's character set.

Z Each character Z is a replacement character that represents a leading numeric position that will be replaced by a space when its contents are zero. Each Z is counted in the size of the item. Its use is explained in Editing Rule 7 below.

9 Each 9 in a picture-string represents a character position that contains a numeral and is counted in the size of the item.

0 Each zero in the picture-string represents a character position into which the numeral 0 will be inserted. The 0 is counted in the size of the item. Its use is explained in Editing Rule 3 below.

/ Each slash mark (/) in the picture-string represents a character position into which the slash character will be inserted. The slash is counted in the size of the item. Its use is explained in Editing Rule 3 below.

, The comma character (,) specifies insertion of a comma between digits. Each such character is counted in the size of the data item, but does not represent a digit position. When DECIMAL-POINT IS COMMA is specified, the explanations for period and comma are reversed to apply to comma and period, respectively. Its use is explained in Editing Rule 3 below.

. A period character (.) in a picture-string is an editing symbol representing the decimal point for alignment purposes. The character also serves to indicate the position for decimal point insertion. Its use is explained in Editing Rule 4 below.

Numeric character positions to the right of an actual decimal point in a PICTURE must consist of characters of one type. The period character (.) is counted in the size of the item. When DECIMAL-POINT IS COMMA is specified, the explanations for period and comma are understood to apply to comma and period, respectively.

The period character must not be the last character in the picture-string.

{+ / - / CR / DB} These symbols are used as editing sign control symbols and represent the character position into which the editing sign control symbol is placed. The symbols are mutually exclusive in any one picture-string, and each character used in the symbol is counted in determining the size of the data item. That is, CR and DB require two character positions each; the + and - require one character position each. Their use is explained in Editing Rules 5 and 6 below.

* Each asterisk in a picture-string is a replacement character. Leading zeros in the affected item are suppressed and replaced by asterisks. Each asterisk is counted in the size of the item. Its use is explained in Editing Rule 7 below.

$ or cs The dollar sign or other currency symbol represents a character position into which the currency symbol is to be placed. Their use is explained in Editing Rule 6 below. The currency symbol is the character specified in the CURRENCY SIGN clause or else the dollar sign. It is counted in the size of the item.

Editing Rules

The rules below define the ways of editing a data field, that is, of adding, changing, or suppressing certain characters in it. Examples are given in Figure 7-3.

1. The maximum length of an edited field is 255 characters. The maximum length of an edited picture-string is 32 characters.

2. The PICTURE clause provides two methods for editing: character insertion and character suppression and replacement.

   There are four types of insertion editing:

   Simple insertion

   Special insertion

   Fixed insertion

   Floating insertion

   There are two types of suppression and replacement editing:

   Zero suppression and replacement with spaces

   Zero suppression and replacement with asterisks

   The type of editing that may be performed upon an item is dependent upon the category to which the item belongs, as defined in Chapter 4. Table 7-2 specifies which type of editing may be performed upon a given category.

Table 7-2
Categories of Data and Editing

Category of Data	Type of Editing Allowed
Alphabetic	Simple insertion (B only)
Numeric	None
Alphanumeric	None
Alphanumeric edited	Simple insertion (0 B and /)
Numeric edited	All, subject to rules for fixed insertion editing

3. Simple insertion editing uses the four symbols B 0 , / as insertion characters. The insertion characters are counted in the size of the item and represent the position in the item into which the character will be inserted. See examples 4-7 for edited fields in Figure 7-3.

4. Special insertion editing refers to decimal point insertion. The period is used as the insertion character. It also represents the decimal point for alignment purposes. The period is counted in the size of the item. The use of the assumed decimal point, represented by the symbol V, and an actual decimal point, represented by the insertion character, in the same picture-string is not allowed. The result of special insertion editing is that the insertion character is placed in an item in the same position where it appears in the picture-string. See examples 8-10 for edited fields in Figure 7-3.

5. Fixed insertion editing employs the currency sign and editing sign control symbols as insertion characters. The four editing sign control symbols are + - CR DB.

   Only one currency symbol, and only one of the editing sign control symbols, can be used in a given picture-string. When the symbols CR or DB are used, they represent two character positions in determining the size of the item. They must represent the rightmost character positions to be counted in the size of the item. The symbol + or -, when used, must be either the leftmost or rightmost character position to be counted in the size of the item. The currency symbol must be the leftmost character position to be counted in the size of an item, except that it can be preceded by either a + or - symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupied in the picture-string. Editing sign control symbols produce the results shown in Table 7-3 depending upon the value of the data item.

Table 7-3
Results of Sign Control Symbols in Editing

Editing Symbol in Picture-string	Result	
	Data Item Positive or Zero	Data Item Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

See examples 11-14 for edited fields in Figure 7-3.

6. Floating insertion editing uses floating insertion characters. These are the currency symbol and the editing sign control symbols + or -. As floating insertion characters, these are mutually exclusive in a given picture-string. These characters cause leading zeros in their positions to be replaced with blanks, except for the leftmost zero, which is replaced with the insertion character.

   A floating string is defined as a leading, continuous series of either $ + or -, or a string composed of one such character interrupted by one or more insertion commas and/or decimal point. Examples are:

```
$$,$$$,$$$
++++
--,---,--
+(8).++
$$,$$$.$$$
```

   Floating insertion editing is indicated in a picture-string by including in it a string of at least two of the floating insertion characters. This floating string may contain any of the fixed insertion symbols. The leftmost character of the floating insertion string represents the leftmost limit of the floating symbol in the data item. The rightmost character of the floating string represents the rightmost limit of the floating symbols in the data item.

   The second floating character from the left represents the leftmost limit of the numeric data that can be stored in the data item. Nonzero numeric data may replace all the characters at or to the right of this limit.

   In a picture-string, there are only two ways of representing floating insertion editing. One way is to represent any or all of the leading numeric character positions on the left of the decimal point by the insertion character. The other way is to represent all of the numeric character positions in the picture-string by the insertion character. See examples 15-16 for edited fields in Figure 7-3.

   If the insertion characters are only to the left of the decimal point in the picture-string, the result is that a single floating insertion character will be placed in the character position immediately preceding the first nonzero digit in the data item. If all data item digits to the left of the decimal are zero, the floating insertion character will be placed in the character position immediately preceding the decimal point. The character positions preceding the insertion character are replaced with spaces.

If all numeric character positions in the picture-string are represented by the insertion character, the result depends on the value of the data. If the value is zero, the entire data item will contain spaces. See examples 17-19 for edited fields in Figure 7-3.

If the value is not zero, the result is the same as when the insertion character is only to the left of the decimal point. See example 18 for edited fields in Figure 7-3.

To avoid truncation, the minimum size of the picture-string for the receiving data item must be the number of characters in the sending data item, plus the number of nonfloating insertion characters being edited into the receiving data item, plus one for the floating insertion character. That is, to define n digit positions, a floating string must contain n + 1 occurrences of $ or + or -.

When a comma appears to the right of a floating string, the comma is not retained if there are no digits retained before it.

Examples:

Picture-string	Numeric Value	Developed Item
$$$999	14	$014
—,—,999	-456	-456
$$$$$$	14	$14

A floating string need not constitute the entire PICTURE of a numeric edited item. However, the characters to the right of a decimal point and up to the end of a PICTURE, excluding the fixed insertion characters +, -, CR, DB (if present), are subject to the following restrictions:

- Only one type of digit position character may appear. That is, the three characters Z * 9 are mutually exclusive, and the floating-string digit position characters $ + - are mutually exclusive.

- If any of the numeric character positions to the right of a decimal point is represented by + or - or $ or Z, then all the numeric character positions in the PICTURE must be represented by the same character.

- Nothing can precede a floating string except + or -.

7. Suppression and replacement editing includes two types: zero suppression and replacement with spaces, and zero suppression and replacement with asterisks.

Floating insertion editing and editing by zero suppression and replacement are mutually exclusive in a PICTURE clause.

The suppression of leading zeros in numeric character positions is indicated by the use of the alphabetic character Z, or the character * (asterisk) as suppression symbols in a picture-string. These symbols are mutually exclusive in a picture-string. Each suppression symbol is counted in determining the size of the item. If Z is used, the replacement character will be the space. If the asterisk is used, the replacement character will be the asterisk. See examples 20-24 for edited fields in Figure 7-3.

Zero suppression and replacement are indicated in a picture-string by one or more of the allowable symbols (Z or *), representing leading numeric character positions. These, in turn, are to be replaced when the associated character position in the data contains a zero. Any of the characters B 0 , / embedded in the string of suppression symbols, or to the immediate right of this string, is part of the string.

There are two ways of representing zero suppression in a picture-string. One way is to represent any or all leading numeric character positions to the left of the decimal point by suppression symbols. The other way is to represent all numeric character positions in the picture-string by suppression symbols.

If the suppression symbols appear only to the left of the decimal point, any leading zero in the data that corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates either at the first non-zero digit in the data represented by the suppression symbol string, or at the decimal point, whichever is first.

If all numeric character positions in the picture-string are represented by suppression symbols, and the value of the data is not zero, the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero, the entire data item will be spaces if the symbol is Z, or all asterisks (except for the actual decimal point) if the symbol is *. See examples 25-27 for edited fields in Figure 7-3.

8. The following symbols can appear only once in a given picture: S V . CR DB.

REDEFINES

Function

The REDEFINES clause allows the same computer storage area to be described by different data-description-entries. It is useful in table handling.

Format

level-number { **data-name-1** / **FILLER** } **[; REDEFINES data-name-2]**

Note

The level-number, data-name-1, and the semicolon are not part of the REDEFINES clause, but are included to show the context.

Syntax Rules

1. The REDEFINES clause is optional; when specified, it must immediately follow data-name-1. The entry for data-name-1 must follow the entry for data-name-2.

2. Level-numbers of data-name-1 and data-name-2 must be identical, but must not be 66 or 88.

3. This clause must not be used in level-number 01 entries in the FILE section.

4. The data-description-entry for data-name-2 may contain a REDEFINES clause.

5. The data-description-entry for data-name-2 may not contain an OCCURS clause, nor may data-name-1 be subordinate to an entry containing an OCCURS clause.

6. The data-name-2 may be qualified but not subscripted.

7. When the level-number of data-name-1 is other than 01, it must specify the same number of character positions as data-name-2.

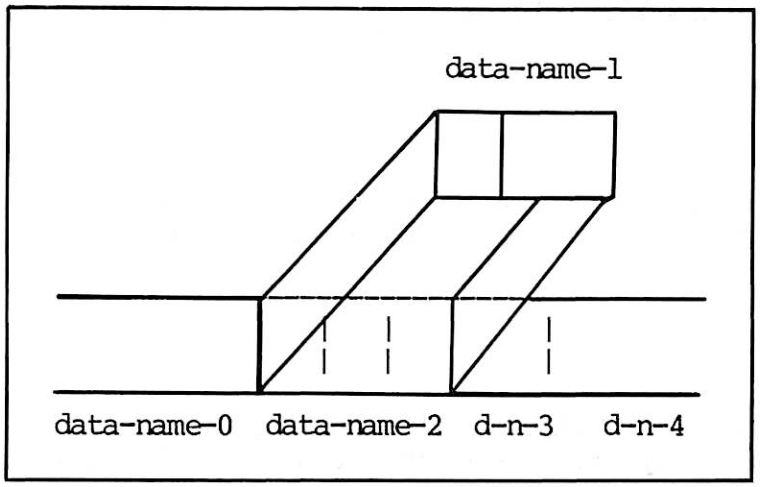
General Rules

1. Redefinition starts at data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered. In

the following example, redefinition of the data-name-2 area by data-name-1 ends when data-name-3 is encountered.

```
05  data-name-2        PICTURE A(3).
05  data-name-1 REDEFINES data-name-2.
    10  ITEM-A         PICTURE A.
    10  ITEM-B         PICTURE AA.
05  data-name-3        PICTURE X.
```

Figure 7-4 represents the projection of these two data-names on one storage area through the REDEFINES clause.



A Storage Area Redefined
Figure 7-4

2. The entries giving the new description of the area must not contain VALUE clauses except in condition-name entries.

3. Redefinition to a depth greater than one level is permitted. (See Syntax Rule 4, above.) Thus, the nested REDEFINES outlined below is valid:

```
01  FIELD-A              PIC X(10).
01  FIELD-B  REDEFINES FIELD-A.
    05  FIELD-C          PIC X(5).
    05  FIELD-D  REDEFINES FIELD-C.
        10  FIELD-E1     PIC X(3).
        10  FIELD-E2     PIC X(2).
    05  FIELD-F          PIC X(5).
```

4. When a value is assigned to one data-name for a redefined storage area, all names for that area will have the same value.

RENAMES

Function

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

Format

```
66 data-name-1;RENAMES data-name-2 [ { THROUGH } data-name-3 ]
                                    { THRU    }
```

Note

The level-number 66, data-name-1, and the semicolon are not part of the RENAMES clause, but are included to show the context.

Syntax Rules

1. Any number of RENAMES entries may be written for a logical record. They must all immediately follow the last entry of that record.

2. The data-name-1 cannot be used as a qualifier but can be qualified by the 01 or FD entries. Neither data-name-2 nor data-name-3 may have an OCCURS clause nor be subordinate to an entry that has an OCCURS clause in its data-description-entry.

3. The data-names 2 and 3 must be the names of elementary items or groups of elementary items in the same record and cannot have the same data-name.

4. A level-66 entry cannot rename another level-66 entry or a 77, 88 or 01 level entry, nor can it rename an entry that contains an 88-level entry.

5. The beginning of the area described by data-name-3 must not be to the left of the beginning of the area described by data-name-2. The end of the area described by data-name-3 must be to the right of the end of the area described by data-name-2. Therefore data-name-3 cannot be subordinate to data-name-2.

6. The data-names 2 and 3 may be qualified.

7. The words THRU and THROUGH are equivalent.

General Rules

1. When data-name-3 is specified, data-name-1 will be a group item that includes all elementary items starting with data-name-2 (if that is an elementary item) or the first elementary item in data-name-2 (if data-name-2 is a group item), and concluding with data-name-3 (if that is an elementary item) or the last elementary item in data-name-3 (if data-name-3 is a group item).

2. When data-name-3 is not specified, data-name-2 can be either a group or an elementary item. When data-name-2 is a group item, data-name-1 is treated as a group item, and when data-name-2 is an elementary item, data-name-1 is treated as an elementary item.

Example

In the following example, the 66-level name VENDOR-ENTRY establishes a new group item from a part of the elements in the record ENTRY-IMAGE. Note that, in this example, RENAMES redefines two level-10 items followed by three level-5 items, while REDEFINES would have redefined only one elementary or one group item.

```
FD  ENTRY-FILE, LABEL RECORDS ARE STANDARD,
      VALUE OF FILE-ID IS 'IN-DATA',
      RECORD CONTAINS 36 CHARACTERS,
      DATA RECORD IS ENTRY-IMAGE.
01  ENTRY-IMAGE.
    05 ENTRY-MONTH.
      10 DY                        PIC 99.
      10 MO                        PIC 99.
      10 YR                        PIC 99.
    05 ENTRY-VENDOR                PIC X(20).
    05 ENTRY-ACCT-NO               PIC 999.
    05 ENTRY-AMOUNT                PIC 9(5)V99.
66 VENDOR-ENTRY      RENAMES MO THRU ENTRY-AMOUNT.
```

SIGN

Function

The SIGN clause specifies the position and the mode of representation of the operational sign when it is necessary to describe these properties explicitly.

Format

$$[\underline{\text{SIGN}}\ \text{IS}] \left\{ \begin{matrix} \underline{\text{LEADING}} \\ \underline{\text{TRAILING}} \end{matrix} \right\} \quad [\underline{\text{SEPARATE}}\ \text{CHARACTER}]$$

Syntax Rules

1. The SIGN clause may be specified only for a numeric data item whose PICTURE contains the character S, or for a group item containing at least one such elementary item. If an S is not present in the data item picture-string, the item is considered unsigned (capable of storing only absolute values), and the SIGN clause is prohibited.

2. Numeric data items to which the SIGN clause applies must be described explicitly or implicitly by USAGE IS DISPLAY.

3. Only one SIGN clause can apply to any given numeric data-description-entry.

4. If the CODE-SET clause is specified in a file-description-entry, any signed numeric item associated with that file must be described with the SIGN IS SEPARATE clause.

General Rules

1. When S appears in a picture-string, but no SIGN clause is included in an item's description, the default is SIGN IS TRAILING.

2. If the optional SEPARATE CHARACTER phrase is not present, then:

   - The operational sign is presumed associated with the leading (or trailing) digit position of the elementary numeric data item.

   - The character S in the picture-string is not counted in determining item size.

3. If the SEPARATE CHARACTER phrase is present, then:

   - The operational sign is presumed to be the leading (or trailing) character position of the elementary numeric data item; this character position is not a digit position.

   - The letter S in a picture-string is counted in determining the size of the item (in terms of standard data format characters).

   - The operational signs for positive and negative are the standard data format characters + and -, respectively.

4. Every numeric data-description-entry whose PICTURE contains the character S is a signed numeric data-description-entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

5. Table 7-4 depicts sign representations for the various SIGN clause options.

Table 7-4
Sign Representation

SIGN Clause	Sign Representation
TRAILING	Embedded in rightmost byte
LEADING	Embedded in leftmost byte
TRAILING SEPARATE	Stored in separate rightmost byte
LEADING SEPARATE	Stored in separate leftmost byte

See the section DATA REPRESENTATION AND ALIGNMENT in Chapter 4 for a detailed description of SIGN formats and conventions.

SYNCHRONIZED

Function

The SYNCHRONIZED clause specifies the alignment of an elementary item on its natural addressing boundaries in the computer memory.

Format

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

Syntax Rules

1. SYNC is an abbreviation for SYNCHRONIZED.

2. In this compiler, the SYNCHRONIZED specification is treated as commentary.

3. Alignment of group items and elementary items of certain data types is done automatically by the compiler. Level-77 and level-01 items are always aligned on halfword (16-bit) boundaries. Group items with level-numbers greater than 01 (subgroups) are aligned on halfwords if they contain fields that require such alignment (COMP, COMP-1, and COMP-2 fields). The data map created with the -MAP compiler option flags items that are aligned by the compiler. In addition, the -SLACKBYTES compiler option causes an observational diagnostic to be issued for each compiler-aligned item.

   See DATA REPRESENTATION AND ALIGNMENT in Chapter 4.

USAGE

Function

The USAGE clause describes the form in which numeric data is represented.

Format

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

Syntax Rules

1. COMP, COMP-1, COMP-2, COMP-3 are abbreviations for COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, and COMPUTATIONAL-3, respectively.

2. COMPUTATIONAL-1, COMPUTATIONAL-2, and COMPUTATIONAL-3 are Prime extensions to ANSI COBOL.

3. The PICTURE clause cannot be used if USAGE is specified as COMPUTATIONAL-1, COMPUTATIONAL-2, or INDEX.

General Rules

1. COMPUTATIONAL defines a binary item. COMPUTATIONAL-1 defines a single-precision floating-point number. COMPUTATIONAL-2 defines a double-precision floating-point number. COMPUTATIONAL-3 specifies a packed decimal item. INDEX defines a binary item to be used for referencing tables. DISPLAY defines an item represented in external decimal format.

   These items and their allowable PICTURE clauses are discussed in the section DATA REPRESENTATION AND ALIGNMENT in Chapter 4. The alignment of COMP, COMP-1, and COMP-2 items is also discussed with the SYNCHRONIZED clause above.

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

3. A COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, or COMPUTATIONAL-3 item can represent a value to be used in computations and must be numeric. When a group item is given one of these usages, only the elementary items in that group may be used in computations.

4. DISPLAY is the system default if the USAGE clause is not specified. Numeric display items may be used in computations.

5. Prime extension: if USAGE is specified as COMPUTATIONAL for an item, and a PICTURE clause is not included for the same item, the compiler assumes a PIC of S9999 (16-bit signed binary integer).

6. COMP-1 and COMP-2 are intended for use in calling certain PRIMOS subroutines that require floating-point (real) arguments. They are also for use in scientific calculations that require a large range at the expense of absolute decimal precision.

   COMP and COMP-3 are for use in most decimal calculations. COMP allows the greatest efficiency both in storage space and in speed of calculations for integers. COMP-3 is more often used to save file and memory space, but it is not an ANSI standard type.

DISPLAY is the only usage allowed for alphabetic characters and symbols. For numbers, it is allowed in calculations but is less efficient than the other usages.

INDEX is allowed only for a value to be used as the index to a table. Allowable operations on the index data type are listed in the discussion of INDEX in Chapter 10.

If any of these data types are mixed together in calculations, such a mixed calculation is allowed, but often at the expense of either precision or efficiency. For details, see the section DATA REPRESENTATION AND ALIGNMENT in Chapter 4.

VALUE

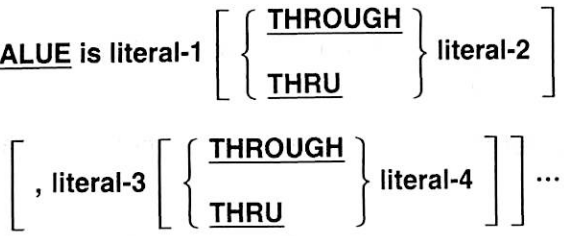
Function

The VALUE clause defines the value of constants, the initial values of WORKING-STORAGE items, and the values associated with a condition-name.

Format 1

VALUE is literal

Format 2



Syntax Rules

1. The words THROUGH and THRU are equivalent.

2. The VALUE clause is not permitted in a data-description-entry specifying an OCCURS or REDEFINES clause, or subordinate to such an entry.

3. A signed numeric literal may be used in a VALUE clause only if the associated picture-string is a signed numeric item.

4. Numeric literals in a VALUE clause must have a value within the range of values indicated by the PICTURE clause, and must not have a value that would have nonzero digits truncated.

   Nonnumeric literals in a VALUE clause must not exceed the size indicated by the PICTURE clause.

5. The type of literal allowed in a VALUE clause depends on the type of data item, as specified in the PICTURE or USAGE clauses. For edited items, values must be specified as nonnumeric literals. A type conflict, producing a compile-time error, will arise if a figurative constant or literal is not compatible with the PICTURE. For example, PICTURE X VALUE 1234 will produce a type conflict error, since 1234 is a numeric figurative constant, but PICTURE X specifies an alphanumeric item.

6. A VALUE clause may not occur in the FILE section of the DATA division except in level-88 condition-name entries.

General Rules

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

2. Initialization takes place independently of any BLANK WHEN ZERO or JUSTIFIED clause that may be specified.

3. The VALUE clause may be specified at the group level in the form of a correctly sized nonnumeric literal, or a figurative constant.

4. A figurative constant may be specified in both Format 1 and Format 2 instead of a literal.

5. Format 1 is required to define an initial value for a data item or a constant.

6. Format 2 is used only for condition-name entries (level-88 items). The VALUE clause and the level-88 condition-name itself are the only two items permitted in the entry. The characteristics of a condition-name are implicitly those of its conditional variable. Wherever the THRU phrase is used, literal-1 must be less than literal-2, literal-3 less than literal-4, etc.

   Level-88 specifications can contain individual values, series of individual values, a range of values, or a series of ranges of values. (See also LEVEL-NUMBER.)

7. Rules governing the VALUE clause differ in the respective sections of the DATA division:

   - In the FILE section, the clause can be used only in condition-name entries.

   - In the WORKING-STORAGE and LINKAGE sections, the clause must be used in condition-name entries. It can also be used in the WORKING-STORAGE section to specify the initial value of any other data item, with the result that the item assumes the specified value at the start of the object program.

   - When an initial value is not specified, no assumption should be made regarding the initial contents of an item in WORKING-STORAGE.

8. If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal. The group area is initialized without consideration for the individual elementary or group items contained within this group. No VALUE clause can be stated at the subordinate levels within this group.

9. The VALUE clause must not be written for a group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE other than USAGE IS DISPLAY.

WORKING-STORAGE SECTION

Function

The WORKING-STORAGE section of the DATA division describes noncontiguous data (level-77 data with no hierarchic relationship) and records that are not part of external files, but are developed and processed internally. Data in this section may be assigned initial values with the VALUE clause.

Format

WORKING-STORAGE SECTION.

[**level-77-description-entry**]
[**record-description-entry**] ...

Syntax Rules

1. The WORKING-STORAGE section is optional. If included, it must begin with the words WORKING-STORAGE SECTION, followed by a period and a space.

2. Noncontiguous item names and record names in the WORKING-STORAGE section must be unique; they cannot be qualified. Subordinate data-names need not be unique if they can be made unique by qualification, or if they are never referenced.

3. The level-number 77 is applied to noncontiguous elementary data items. Each noncontiguous item must be defined in a separate data-description-entry. The following data clauses are required in each noncontiguous data-description-entry:

   - level-number 77
   - data-name
   - The PICTURE clause or the USAGE IS INDEX, COMP, COMP-1, COMP-2, or COMP-3 clause

   Other data description clauses are optional and can be used to complete the description of the item if necessary.

4. Data items in the WORKING-STORAGE section with a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of record descriptions. Any clause used in a record description in the FILE section can be used in a record description in the WORKING-STORAGE section. (See RECORD-DESCRIPTION-ENTRY above.)

General Rules

1. WORKING-STORAGE items described in this chapter include the following:

   - Noncontiguous elementary items with the level-number 77. These items have no hierarchical relationship to one another and cannot be grouped into records or further subdivided.

   - Data items in records not associated with an input-output device and not part of external data files, but developed and processed internally. These items employ level-numbers 01 through 49.

2. VALUE clauses, prohibited in the FILE section, are permitted throughout WORKING-STORAGE to specify the initial value of an item, except for an index data item.

LINKAGE SECTION

Function

The LINKAGE section describes data previously defined in a calling program which is available to a called program.

Format

LINKAGE SECTION.

[**level-77-description-entry**]
[**record-description-entry**] ...

Syntax Rules

1. The LINKAGE section is optional. It is meaningful only in a called program. If included, it must begin with the words LINKAGE SECTION followed by a period and a space.

2. Each LINKAGE section record-name and noncontiguous item name must be unique within the called program; they cannot be qualified.

3. The level-number 77 refers to noncontiguous elementary data items. Each level-number 77 data item is defined in a separate data-description-entry. The following data clauses are required in each noncontiguous data-description-entry:

   - level-number 77
   - data-name
   - The PICTURE clause or the USAGE IS INDEX, COMP, COMP-1, COMP-2, or COMP-3 clause

   Other data description clauses are optional and can be used to complete the description of the item if necessary.

4. Data items in the LINKAGE section which have a definite hierarchic relationship to one another must be grouped into records according to the rules for record-description-entries.

5. Items in the LINKAGE section cannot have initial values unless they have a level-number of 88.

General Rules

1. The LINKAGE section of the DATA division is meaningful only if the program containing it is called by another program whose CALL statement contains a USING phrase.

2. The LINKAGE section is used to describe data that is available through the calling program, and is to be referred to in both the calling program and the called program. No space is allocated in the program for data items defined in the LINKAGE section of that program. PROCEDURE division references to these data items are resolved at runtime by equating the reference in the called program to the location used in the calling program.

3. Data items defined in the LINKAGE section of the called program may be used within the PROCEDURE division of the called program only if they are specified as operands of the USING phrase of the PROCEDURE division header, or are subordinate to such operands, and the object program is under the control of a CALL statement which specifies a USING phrase.

Note

A LINKAGE section example is presented in Chapter 9, INTERPROGRAM COMMUNICATION.

EXAMPLE

This DATA division listing forms one program with the examples at the end of Chapters 5, 6, and 8.

```
 DATA DIVISION.
*
 FILE SECTION.
*
 FD  DISK-FILE COMPRESSED,
     LABEL RECORDS ARE STANDARD,
     VALUE OF FILE-ID IS 'DISBURSE',
     RECORD CONTAINS 42,
     DATA RECORD IS ENTRY-DETAIL.
 01  ENTRY-DETAIL.
     05 ENTRY-CHECK-NO          PIC X(3).
     05 ENTRY-MONTH.
        10 ENTRY-MM             PIC 99.
        10 ENTRY-DD             PIC 99.
        10 ENTRY-YY             PIC 99.
     05  FILLER                 PIC XXX.
     05  ENTRY-VENDOR           PIC X(20).
     05  ENTRY-ACCT-NO          PIC 999.
     05  ENTRY-AMOUNT           PIC 9(5)V99.
/
 FD  PRINT-FILE,
     LABEL RECORDS ARE OMITTED,
     DATA RECORDS ARE PRINT-LINE, ERROR-LINE.
 01  PRINT-LINE                 PIC X(70).
 01  ERROR-LINE.
     05 MESSAGE                 PIC X(20).
     05 ERR-CODE                PIC 9.
*
 FD  TAPE-FILE,
     LABEL RECORD IS STANDARD,
     BLOCK CONTAINS 4 RECORDS,
     VALUE OF FILE-ID IS TAPENAME,
     DATA RECORD IS TAPE-LINE.
*
 01  TAPE-LINE                  PIC X(20).
*
 WORKING-STORAGE SECTION.
 77  BLANKS                     PIC X(2)              VALUE ZERO.
 77  CROSS-TOTAL                PIC S9(8)V99  COMP-3  VALUE ZERO.
 77  FINAL-TOTAL                PIC S9(8)V99  COMP-3  VALUE ZERO.
 77  GRAND-TOTAL                PIC S9(8)V99  COMP-3  VALUE ZERO.
 77  JOB-DATE                   PIC 9(6)              VALUE ZERO.
 77  LIMIT-DATE                 PIC S9(5).
 77  LINECOUNT                  PIC S99       COMP-3  VALUE ZERO.
 77  NO-MORE-RECORDS            PIC X                 VALUE 'N'.
 77  PAGECOUNT                  PIC S9        COMP-3  VALUE 1.
 77  REJECT-TOTAL               PIC S9(7)V99  COMP-3  VALUE ZERO.
 77  TAPE-CHOICE                PIC XXX               VALUE 'NO '.
 77  TAPENAME                   PIC X(20)
```

```
          VALUE IS '$MT0, S, ANNE, T1'.
 77  TOTAL1                  PIC S9(7)V99    COMP-3 VALUE ZERO.
 77  TOTAL2                  PIC S9(7)V99    COMP-3 VALUE ZERO.
 77  TOTAL3                  PIC S9(7)V99    COMP-3 VALUE ZERO.
 77  TOTAL4                  PIC S9(7)V99    COMP-3 VALUE ZERO.
 77  TOTAL5                  PIC S9(7)V99    COMP-3 VALUE ZERO.
 77  TOTAL6                  PIC S9(7)V99    COMP-3 VALUE ZERO.
 77        VARIABLE               PIC S9          VALUE 2.
*JOB-INFO IS ACCEPTED FROM CONSOLE:
 01  JOB-INFO.
     03 JOB-CODE             PIC S99.
       88 CORRECT-CODE       VALUE 25.
*****************************************************************
*                PRINT-LINES
*****************************************************************
 01  HEADING1.
     03 CARRIAGE-CONTROL          PIC X.
     03 FILLER                    PIC X(18)      VALUE SPACES.
     03 FILLER                    PIC X(34)
          VALUE 'MONTHLY CASH DISBURSEMENTS JOURNAL'.
     03 FILLER                    PIC X(12)      VALUE SPACES.
 01  HEADING2.
     03 CARRIAGE-CONTROL          PIC X.
     03 FILLER                    PIC X(25)      VALUE SPACES.
     03 FILLER                    PIC X(7)       VALUE 'FOR    '.
     03 VARIABLE-MONTH            PIC X(15).
     03 FILLER                    PIC X(15)      VALUE SPACES.
     03 FILLER                    PIC X(4)       VALUE 'PAGE'.
     03 HEADING-PAGE              PIC ZZ9.
 01  HEADING3.
     03 CARRIAGE-CONTROL          PIC X.
     03 FILLER                    PIC X(24)      VALUE SPACES.
     03 VARIABLE-HEADING          PIC X(24)      VALUE SPACES.
     03 FILLER                    PIC X          VALUE SPACES.
 01  PRINT-DETAIL.
     05 FILLER                    PIC X(1)       VALUE SPACES.
     05 ENTRY-MONTH               PIC 9(6).
     05 FILLER                    PIC X(6)       VALUE SPACES.
     05 ENTRY-VENDOR              PIC X(20).
     05 FILLER                    PIC X(7)       VALUE SPACES.
     05 ENTRY-CHECK-NO            PIC X(3).
     05 FILLER                    PIC X(9)       VALUE SPACES.
     05 PRINT-ACCT-NO             PIC X(3).
     05 FILLER                    PIC X(6)       VALUE SPACES.
     05 PRINT-AMOUNT              PIC ZZZZZZ.99.
 01  HOME-ACCT-LINE.
     05 FILLER                    PIC X(13)      VALUE SPACES.
     05 HOME-NUMBER               PIC X(21).
     05 HOME-TOTAL                PIC Z(8).99.
 01  BALANCE-LINE.
     05 FILLER                    PIC X(9)       VALUE SPACES.
     05 FIELD-TOTAL               PIC Z(8).99.
     05 FILLER                    PIC X(8)       VALUE SPACES.
     05 FIELD-REJECT              PIC Z(8).99.
```

```
     05 FILLER                    PIC X(9)       VALUE SPACES.
     05 FIELD-DIFF                PIC Z(8).99.
     05 FILLER                    PIC X(11)      VALUE SPACES.
*****************************************************************
*       TAPE OUTPUT
*****************************************************************
 01  TAPE-HEADER.
     05 TAPE-MONTH                PIC X(15)      VALUE SPACES.
     05 FILLER                    PIC X(5)       VALUE SPACES.
 01  SAVE-TAPE.
     05 SAVE-DATE-TAPE            PIC 9(6).
     05 SAVE-ACCT-TAPE            PIC XXX.
     05 SAVE-TOTAL-TAPE           PIC S9(9)V99     COMP-3.
     EJECT
*************************************************************
```

8
The PROCEDURE DIVISION

PROCEDURE DIVISION

Function

The PROCEDURE division contains instructions specifying the steps to be performed by the program. COBOL instructions are written as statements or sentences that may be combined to form paragraphs headed by paragraph-names. These, in turn, may be combined to form sections headed by section-names.

Format 1

```
PROCEDURE DIVISION [ USING data-name-1 [, data-name-2] ··· ] .

[ DECLARATIVES.

  { section-name SECTION [segment-number]. USE-sentence.
  { [paragraph-name. [sentence] ··· ] ···                  } ...

  END DECLARATIVES. ]

[section-name SECTION [segment-number]. ]···

[paragraph-name. [sentence] ··· ] ···
```

Format 2

PROCEDURE DIVISION [USING data-name-1 [, data-name-2] ···] .

[section-name SECTION [segment-number].]

[paragraph-name. [sentence] ···] ···

Syntax Rules

1. The first entry in the PROCEDURE division must be the words PROCEDURE DIVISION, followed by a period and a space unless the USING clause is included.

2. The USING clause is specified only if both of the following occur:

   - The program being written is a CALLable subprogram that is to function under the control of a CALL statement.
   - The CALL statement in the calling program contains a USING clause.

3. Each of the data-name operands in the USING clause must be defined as a data item in the LINKAGE section of the program.

4. Within the program, LINKAGE section data items are processed according to their data descriptions given in the program itself.

5. Prime extension: level-numbers of data-names in the USING clause may be 01 through 49 or 77. See Chapter 9, INTERPROGRAM COMMUNICATION, for a complete discussion.

6. Declarative sections are optional. When included, they must be grouped at the beginning of the PROCEDURE division, preceded by the keyword DECLARATIVES and followed by the keywords END DECLARATIVES. These entries must appear on separate lines.

7. A declarative section must have a section header. When included, it must consist of a section-name, followed by the word SECTION and a period. Each section-name must appear on a line by itself; each section-name must be unique.

8. A paragraph is an entry consisting of zero or more sentences, preceded by a paragraph-name.

9. Segment-numbers are included only for compatibility. They must be integers from 0 to 99.

10. Paragraph-names and section-names follow the general rules for word formation in Chapter 4. These names may be all numeric.

11. A sentence is a single statement or a series of statements terminated by a period and followed by a space.

12. A statement consists of a COBOL verb followed by appropriate operands (data-names or literals) and other clauses necessary for the completion of the statement. Statements are classed as imperative, conditional, or compiler-directing.

    - An imperative statement specifies an unconditional action to be taken by the object program. An imperative statement consists of a verb and its operands, excluding the IF conditional statement, the READ statement, any arithmetic statement with the SIZE ERROR clause, and any I-O statement with an INVALID KEY or AT END clause.

    - A conditional statement stipulates a condition that is tested to determine whether an alternate path of program flow is to be taken. IF statements, arithmetic statements with SIZE ERROR, and any I-O statement having an INVALID KEY or AT END clause are conditional. See Chapter 4 for rules governing conditional statements.

    - Compiler-directing statements cause the compiler to perform an action but have no effect on execution of the object program. USE, EJECT, SKIP, and COPY are directives to the compiler.

13. The maximum code size in the PROCEDURE division is listed in Appendix J.

Arithmetic Statements in the PROCEDURE Division

1. Arithmetic statements may be imperative or conditional. The five arithmetic verbs are: ADD, SUBTRACT, MULTIPLY, DIVIDE, COMPUTE. Arithmetic statements in the PROCEDURE division are governed by the following rules:

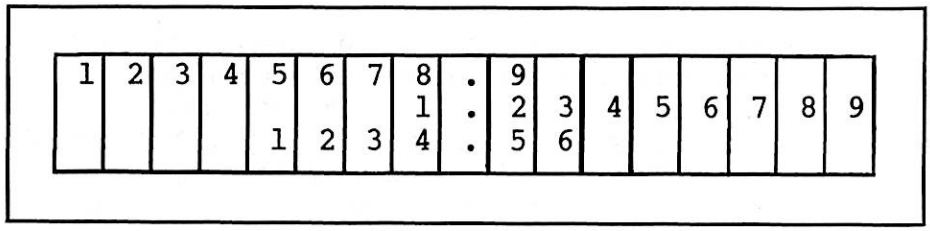
    - All data-names used in arithmetic statements must be elementary numeric data items defined in the DATA division of the program, except when they are the operands of GIVING, in which case they may be numeric edited. Index-names and index items are not permitted in these arithmetic statements.

    - Decimal-point alignment is supplied automatically throughout the computations.

- The maximum size of each operand is 18 decimal digits.

- The composite of operands for an arithmetic statement is a hypothetical data item resulting from superimposition of its operands aligned on their decimal points. For example, 12345678.9 and 1.23456789, superimposed, form a composite having 16 digits. No composite of operands may contain more than 18 decimal digits unless it is used with COMPUTE or the GIVING option.

  The composite of the operands 12345678.9, 1.23456789, and 1234.56 may be pictured as in Figure 8-1.



Composite of Operands
Figure 8-1

2. The four clauses that may appear in arithmetic statements are: the GIVING option, the ROUNDED option, the SIZE ERROR option, and the CORRESPONDING clause. GIVING may not be used with COMPUTE.

   - If the GIVING option is used, the value of the data-name that follows the word GIVING is made equal to the calculated result of the arithmetic operation. The data-name that follows GIVING is not used in the computation and may be a numeric edited item.

   - When the ROUNDED option is specified, if the most significant digit of the excess is greater than or equal to 5, the least significant digit of the resultant data-name has its value increased by 1. If the ROUNDED option is not used, truncation and, hence, loss of precision may occur.

     Rounding of a computed negative result is performed by rounding the absolute value of the computed result and then making the final result negative.

     Figure 8-2 illustrates the relationship between a calculated result and the value stored in an item that is to receive the calculated result, with and without rounding.

Calculated Result	Item to Receive Calculated Result		
	PICTURE	Value After Rounding	Value After Truncating
-12.36	S99V9	-12.4	-12.3
8.432	9V9	8.4	8.4
35.6	99V9	35.6	35.6
65.6	S99V	66	65
.0055	SV999	.006	.005

Rounding Results
Figure 8-2

- The SIZE ERROR option is written immediately after any arithmetic or numeric MOVE statement, as an extension of the statement. The format of the SIZE ERROR option is:

  [ON SIZE ERROR imperative-statement(s) ...]

  If, after decimal-point alignment, the absolute value of a calculated result exceeds the largest value that the receiving field is capable of holding, a size error condition exists.

  Division by 0 always causes a size error condition. The size error condition applies only to the final results of an arithmetic operation and does not apply to intermediate results. If ROUNDED is specified, rounding takes place before checking for size errors.

  If the SIZE ERROR option is present, and a size error condition arises, the value of the receiving data-name is unaltered and the series of imperative statements specified for the condition is executed.

  If the SIZE ERROR option has not been specified for arithmetic statements and a size error condition arises, the final result is undefined. Truncation usually results.

  An arithmetic or numeric MOVE statement, if written with a SIZE ERROR option, is a conditional statement since it is data-dependent. It is therefore prohibited in contexts where only imperative statements are allowed.

  An example of a conditional arithmetic statement is:

```
ADD 1 TO RECORD-COUNT, ON SIZE ERROR MOVE ZERO TO
RECORD-COUNT, DISPLAY "LIMIT 99 EXCEEDED".
```

If RECORD-COUNT has PICTURE 99, and has the value 99, it cannot be incremented, so both the MOVE and DISPLAY statements are executed. Otherwise, the MOVE and DISPLAY statements are not executed.

- The CORRESPONDING clause may be used with all arithmetic statements, plus MOVE and IF, in Prime COBOL. It requires two operands, group-1 and group-2, which must each refer to group items. A pair of data items, one from group-1 and one from group-2, correspond if the following three conditions exist.

  A data item in group-1 and a data item in group-2 are not designated by the keyword FILLER and have the same data-name and the same qualifiers up to, but not including, group-1 and group-2.

  At least one of the data items is an elementary data item in the case of a MOVE statement. Both of the data items are elementary numeric data items in the case of the arithmetic statements. In IF statements, both items may be elementary or nonelementary.

  The description of group-1 and group-2 does not contain level-number 66, 77, or 88 or the USAGE IS INDEX clause.

  A data item that is subordinate to group-1 or group-2 and contains a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause is ignored, as well as any items subordinate to it. However, group-1 and group-2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses.

- If ROUNDED is specified with the CORRESPONDING option, rounding as described above is performed on each matched receiving operand.

- If SIZE ERROR is specified in conjunction with the CORRESPONDING option, a size error test is made for each pair of operands that are matched under the rules for CORRESPONDING, irrespective of the results of any previous SIZE ERROR calculations in the statement. The imperative statement specified as the SIZE ERROR option is executed if any matching pairs of operands have caused a size error.

DECLARATIVE STATEMENTS

The sections under the DECLARATIVES header provide procedures that are invoked when an I-O condition occurs that is not otherwise provided for by the program.

A declarative section may handle errors for multiple statements referring to one file, in place of several AT END or INVALID KEY clauses. In addition, only a declarative section can cover the multiuser situation where a record accessed by one program has already been locked by another program.

Since such procedures are executed only at the time an error in reading and writing occurs, they cannot appear in the regular sequence of procedural statements. Instead, they must appear in the DECLARATIVES section. Error-handling procedures for each file must be grouped together and preceded by a separate USE sentence.

For additional information, see the USE statement in this chapter. An example is given at the end of Chapter 8.

Syntax Rules

1. Each declarative section includes a section header, a USE sentence, and, optionally, one or more paragraphs.

2. END DECLARATIVES must be followed by a period.

3. See the USE statement in this chapter for more information on the DECLARATIVES section.

PROCEDURE STATEMENTS

COBOL statements are described on the following pages in alphabetic order. For a list of statements and other reserved words, see the reserved words list, Table A-2 in Appendix A.

ACCEPT

Function

The ACCEPT statement causes low-volume data to be moved to the specified data-name.

Format 1

ACCEPT data-name [FROM mnemonic-name]

Format 2

ACCEPT data-name FROM { **DATE** / **DAY** / **TIME** }

Syntax Rule

The mnemonic-name in Format 1 must be specified in the SPECIAL-NAMES paragraph of the ENVIRONMENT division, with the CONSOLE IS clause.

Only one transfer of data, and therefore only one data-name, is allowed after ACCEPT.

General Rules

1. The ACCEPT statement causes transfer of data from the terminal or system clock. The transferred data replaces the contents of the field specified by data-name.

2. One line is read, and as many characters as necessary (depending on the size of the named data field) are moved to the indicated field. The maximum number of characters that can be read is 256.

3. When input is to be accepted from the terminal, execution consists of the following steps:

   - Execution is suspended.
   - When the user enters a carriage return, the program stores the keyed-in data preceding the carriage return in the field designated by data-name, and normal execution proceeds.

- For unequal sizes of data-name and terminal input the result is treated as an alphanumeric to alphanumeric move with space-fill on the right or right truncation.

- A single line as long as 256 characters may be transferred.

4. The Format-2 ACCEPT statement causes the requested information to be transferred to the data item specified by data-name according to the rules of the MOVE statement. DATE, DAY, and TIME are reserved words and should not be described in the COBOL program.

5. DATE has the following data elements: year, month, and day of the month, in that sequence. Thus July 1, 1974 is expressed as 740701. DATE, when accessed by a COBOL program, is treated as though described in the COBOL program as an unsigned elementary numeric integer data item six digits long.

6. DAY has the following data elements: year and day of year, in that sequence. July 1, 1974 would be expressed as 74183. DAY, when accessed by a COBOL program, is treated as though described in the COBOL program as an unsigned elementary numeric integer data item five digits long.

7. TIME has the following data elements: hours, minutes, seconds, and hundredths of a second. TIME is based on time elapsed after midnight on a 24-hour basis; thus 2:41 p.m., or 1441 hours, is expressed as 14410000. TIME, when accessed by a COBOL program, is treated as though described in the COBOL program as an unsigned elementary numeric integer data item eight digits long. The minimum value of TIME is 00000000; maximum value is 23595999.

Note

ACCEPTed data does not perform functions declared in the definition of the variable such as BLANK WHEN ZERO or JUSTIFIED. All input, including numbers, is left-justified. When accepting numbers for calculations, use UNSTRING after moving the input to a JUSTIFIED RIGHT field before doing calculations. The following code is an example. Another example is provided with UNSTRING below.

Example

```
       ID DIVISION.
       PROGRAM-ID. CALC.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 DISPLAY-TOTAL                    PIC X(8).
       01 WORK-TOTAL                       PIC X(8) JUSTIFIED RIGHT.
       01 TOTAL-WORK                       PIC S9(6)V99.
      *
       PROCEDURE DIVISION.
       000-INITIALIZE.
           DISPLAY 'WHAT IS INITIAL VALUE OF TOTAL?'.
           DISPLAY '     ** NOTE FORMAT MUST NOT USE DECIMAL POINT.'
           DISPLAY '     ** EX: TO REGISTER $45.25, ENTER 4525.'.
           ACCEPT DISPLAY-TOTAL.
           UNSTRING DISPLAY-TOTAL DELIMITED BY SPACE INTO WORK-TOTAL.
           MOVE WORK-TOTAL TO TOTAL-WORK.
           DIVIDE 100 INTO TOTAL-WORK.
```

ADD

Function

The ADD statement adds together two or more numeric values and stores the resulting sum.

Format 1

```
ADD { data-name-1 } [ , data-name-2 ]
    { literal-1   } [ , literal-2   ] ... TO data-name-3 [ROUNDED]
    { arith-expr-1} [ , arith-expr-2]

    [, data-name-n [ROUNDED]] ...

    [; ON SIZE ERROR imperative-statement]
```

Format 2

```
ADD { data-name-1 } { , data-name-2 } [ , data-name-3 ]
    { literal-1   } { , literal-2   } [ , literal-3   ] ...
    { arith-expr-1} { , arith-expr-2} [ , arith-expr-3]

    GIVING data-name-4 [ROUNDED][, data-name-n [ROUNDED]] ...

    [; ON SIZE ERROR imperative-statement]
```

Format 3

```
ADD { CORRESPONDING } data-name-1 TO data-name-2 [ROUNDED]
    { CORR          }

    [; ON SIZE ERROR imperative-statement]
```

Syntax Rules

1. In Formats 1 and 2, each data-name must refer to an elementary numeric item, except that in Format 2 each item following GIVING can be either an elementary numeric item or an elementary numeric edited item.

2. Each literal must be a numeric literal.

3. The maximum size of each operand is 18 digits. If all operands, excluding those following the word GIVING, were to be superimposed upon each other, aligned by their implied decimal points, their composite could not exceed 18 decimal digits in length.

4. In Format 3, elementary items subordinate to group-name-1 are added to and stored into the corresponding elementary items subordinate to group-name-2. Here, data-name-1 and data-name-3 must be group items.

5. The use of arithmetic expressions in ADD statements is a Prime extension.

General Rules

1. In Format 1, the values of the operands preceding the word TO are added, the sum is added to the current value of data-name-3 and the result is stored in data-name-3. This process is repeated for each operand following TO.

2. In Format 2, the values of the operands preceding the word GIVING are added, and the sum is stored as the new value of data-name-4.

3. In Format 3, data items in data-name-1 are added to and stored in corresponding data items in data-name-2.

4. The ON SIZE ERROR option should be used when truncation of the results could occur.

5. The rules for signs are those presented in the section on ALGEBRAIC SIGNS in Chapter 4.

6. The ADD statement is governed by the rules for GIVING, ROUNDED, SIZE ERROR, and CORRESPONDING in Arithmetic Statements in the PROCEDURE Division at the start of this chapter, and by the rules for Arithmetic Statements in Chapter 4.

Examples

```
ADD INTEREST, DEPOSIT TO BALANCE ROUNDED.
ADD REGULAR-TIME, OVERTIME GIVING GROSS-PAY.
ADD CORRESPONDING DETAIL-LINE TO TOTAL-LINE.
```

The first statement would result in the total sum of INTEREST, DEPOSIT, and BALANCE being rounded and placed in BALANCE, while the second would result in the sum of REGULAR-TIME and OVERTIME being placed in the item GROSS-PAY. The third statement causes elementary items in DETAIL-LINE to be added to items of the same name in TOTAL-LINE.

ALTER

Function

The ALTER statement modifies a simple GO TO statement elsewhere in the PROCEDURE division, thus changing the sequence of execution of program statements.

Format

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

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

Syntax Rules

1. The procedure-names 1, 3, and so on contain a single GO TO sentence without the DEPENDING clause.

2. The procedure-names 2, 4, and so on name other paragraphs or sections in the PROCEDURE division.

General Rule

Execution of the ALTER statement modifies the GO TO statement of the first procedure-name so that subsequent executions of the modified GO TO statement cause transfer of control to the second procedure-name.

CALL

Function

The CALL statement allows one program to communicate with one or more other programs. It causes control to be transferred from one object program to another within a runfile, with both programs having access to data items referred to in the CALL statement.

Format

CALL literal-1 [USING data-name-1 [, data-name-2] ···]

[; ON OVERFLOW imperative-statement]

The CALL statement is presented in detail in Chapter 9, INTERPROGRAM COMMUNICATION.

CLOSE

Function

The CLOSE statement terminates the processing of files.

Format

CLOSE file-name-1 [, file-name-2] ···

Syntax Rule

The files referenced in the CLOSE statement need not all have the same access or organization.

General Rules

1. A CLOSE statement should be executed before a STOP RUN is executed, if any files were opened by the program.
2. A CLOSE statement implies a preceding OPEN on the same file.

COMPUTE

Function

The COMPUTE statement evaluates an arithmetic expression and then stores the result in a designated item.

Format 1

```
COMPUTE data-name-1 [ROUNDED]
  [, data-name-2 [ROUNDED]] ... = arith expr
  [; ON SIZE ERROR imperative-statement]
```

Format 2

```
COMPUTE { CORRESPONDING } data-name-1 [ROUNDED] = data-name-2
        { CORR          }
  [; ON SIZE ERROR imperative-statement
```

Syntax Rules

1. In Format 1, data-names appearing to the left of the equals sign must refer to either an elementary numeric item or an elementary numeric edited item.

2. In Format 2, data-name-1 and data-name-2 must be group items.

General Rules

1. The COMPUTE statement is governed by the rules for ROUNDED, SIZE ERROR, and CORRESPONDING in Arithmetic Statements in the PROCEDURE Division at the start of this chapter. It is also governed by the general rules for ARITHMETIC EXPRESSIONS described in Chapter 4.

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

3. In Format 1, an arithmetic expression may consist of a single data-name or literal. It provides a method of setting the values of data-name-1, data-name-2, and so on, equal to the value of the arithmetic expression.

4. In Format 1, if more than one data-name precedes the equals sign, the value of the arithmetic expression is computed, and then this value is stored as the new value of each data-name.

5. In Format 2, data items in data-name-1 are set to the contents of matching data items in data-name-2.

COPY

Function

The COPY statement incorporates COBOL source coding from another file into a source program at compile time. This is a compiler-directing function.

Format 1

```
COPY 'file-name' [ { OF } 'UFD-name' ]
                 { IN }

[ REPLACING { , { ==pseudo-text-1== } } BY { ==pseudo-text-2== } } ... ]
            {   { data-name-1       }     { data-name-2       }
            {   { literal-1         }     { literal-2         }
            {   { reserved-word-1   }     { reserved-word-2   }
```

Format 2

```
COPY literal-1 [ { OF } literal-2 ]
               { IN }

[ REPLACING { , { ==pseudo-text-1== } } BY { ==pseudo-text-2== } } ... ]
            {   { data-name-1       }     { data-name-2       }
            {   { literal-3         }     { literal-4         }
            {   { reserved-word-1   }     { reserved-word-2   }
```

Syntax Rules

1. OF and IN are interchangeable and mutually exclusive.

2. A COPY statement may occur anywhere in the source program, in any division where a character-string or a separator might usually occur, except that it may not occur within the object of another COPY statement, or in a comment-entry. COPY copies everything in the text-file.

3. Pseudo-text is a literal string, with no quote marks unless these quote marks are to appear in the text. Pseudo-text allows quotes to be copied.

General Rules

1. The file-name must be the name of a PRIMOS file containing COBOL source code.

2. The UFD-name must be the UFD name that contains the file-name. If no UFD-name is given, the UFD of the source-program is assumed.

   Of the examples below, the first and second ones copy files contained on the same UFD as the source program. The third and fourth copy files contained in a UFD named SUB. In the last example, the UFD-name contains a period, which has a special significance in COBOL; to avoid an error, the UFD-name is enclosed in quotes.

```
FILE-CONTROL.  COPY file-name.
SECTION-NAME SECTION.  COPY file-name.
FD MASTER-FILE COPY file-name OF SUB.
01 MASTER-RECORD.  COPY file-name IN SUB.
PARAGRAPH-NAME.  COPY file-name IN 'ANNE.F'.
```

3. Literals 1 and 2 are nonnumeric literals. The literal-3 must be a PRIMOS file-name. The literal-4 must be a UFD-name.

Example

The following is from DATA division coding in a source program.

```
01  MASTER-DESCRIPTION.  COPY MASDES
                         REPLACING ==03== BY ==05==
                                ==PIC X(15)== BY ==PIC X(20)==
01  EMPLOYMENT-HISTORY.
```

The file MASDES must be in the same UFD as the source program. It must not contain the 01 MASTER-DESCRIPTION entry; it might have the format:

```
03  BADGE-NO  PIC 9(5).
03  NAME.
    10  LAST-NAME  PIC X(15).
    10  FIRST-NAME  PIC X(15).
```

After compilation, the listing file would include the following:

```
   60          .
   61          .
   62   01  MASTER-DESCRIPTION.  COPY MASDES
   63                             REPLACING ==03== BY ==05==
   64                                       ==PIC X(15)== BY ==PIC X(20)==
<     1>     05  BADGE-NO  PIC 9(5).
<     2>     05  NAME.
<     3>         10  LAST-NAME  PIC X(20).
<     4>         10  FIRST-NAME  PIC X(20).
   65   01  EMPLOYMENT-HISTORY.
   66          .
   67          .
```

In this example, the COPY statement part of lines 62-64 is a comment only. Line numbering of the inserted text is independent of the line numbers of the source.

DELETE

Function

The DELETE statement removes a record from an indexed or relative file.

Format

DELETE file-name RECORD [; INVALID KEY imperative-statement]

DELETE is used only for indexed and relative files. It is discussed in Chapters 12 and 13 of this guide.

DISPLAY

Function

The DISPLAY statement causes low-volume data to be output to the console.

Format

```
DISPLAY { data-name-1 } [ , data-name-2 ] ... [ UPON mnemonic-name]
        { literal-1   }   { literal-2   }

[[WITH] NO ADVANCING ]
```

Syntax Rules

1. The mnemonic-name must be specified in the CONSOLE IS clause of the SPECIAL-NAMES paragraph in the ENVIRONMENT division.

2. The maximum number of characters that may be output is 256 per DISPLAY statement. More causes truncation.

3. Display items are left-justified (truncated on the right).

General Rules

1. When the UPON clause is omitted, the system default is the terminal.

2. If a figurative-constant is given as an operand, it will be displayed as a single character.

3. Prime extension: data whose usage is not DISPLAY is converted to trailing separate sign format and displayed as shown in Table 8-1. The size of the displayed data item, in characters, is the number of P's plus the number of 9's in the PICTURE clause, plus 3.

4. Prime extension: NO ADVANCING displays a line of information to the terminal but does not output a carriage return. It thus allows, for example, an answer to be input from the terminal on the same line with the question.

Examples

Type	Statement	Output
data-name (numeric)	DISPLAY BADGE-NO	52207
data-name (nonnumeric)	DISPLAY NAME	JOHN DOE
literal	DISPLAY 'END-JOB'	END-JOB
figurative-constant	DISPLAY ZERO	0
COMP	DISPLAY COMPTEST	9
COMP-1	DISPLAY FLOAT-1	3.230000E+03
COMP-2	DISPLAY FLOAT-2	-3.2300000000000E+0003
COMP-3	DISPLAY COMPTHREE	-1
INDEX	DISPLAY TESTINDEX	(blank)

Table 8-1
DISPLAY of Binary Data Types

(After Conversion, If Necessary, to Display Type)

Original Data Type	Size of Display Item* in Characters (Bytes)
Signed COMP — 16 bits	9
— 32 bits	14
— 64 bits	22
Unsigned COMP — 16 bits	9
— 32 bits	14
— 64 bits	22
COMP-1	14
COMP-2	23

* If the data item is a group item, then the item displayed is treated as alphanumeric with a size equal to the total number of bytes in the group.

DIVIDE

Function

The DIVIDE statement divides one numeric data item into another and stores the quotient and, optionally, remainder.

Format 1

```
DIVIDE { data-name-1 }
       { literal-1   } INTO data-name-2 [ROUNDED]
       { arith-expr-1}

  [, data-name-3 [ROUNDED]] ...

  [; ON SIZE ERROR imperative-statement]
```

Format 2

```
DIVIDE { data-name-1 } { INTO } { data-name-2 }
       { literal-1   } {      } { literal-2   }
       { arith-expr-1} { BY   } { arith-expr-2}

  GIVING data-name-3 [ROUNDED]

  [, data-name-4 [ROUNDED]] ...

  [; ON SIZE ERROR imperative-statement]
```

Format 3

```
DIVIDE { data-name-1 } { INTO } { data-name-2 }
       { literal-1   } {      } { literal-2   }
       { arith-expr-1} { BY   } { arith-expr-2}

  GIVING data-name-3 [ROUNDED]

  REMAINDER data-name-4 [; ON SIZE ERROR imperative-statement]
```

Format 4

```
DIVIDE { CORRESPONDING } data-name-1 { INTO } data-name-2 [ROUNDED]
       { CORR          }             { BY   }

  [; ON SIZE ERROR imperative-statement]
```

Syntax Rules

1. Each data-name must refer to an elementary numeric item, except that a data-name associated with the GIVING or REMAINDER phrase can refer either to an elementary numeric item or to an elementary numeric edited item.

2. Each literal must be a numeric literal.

3. The maximum size of each operand is 18 decimal digits. If all receiving data items were to be superimposed upon each other, aligned by their decimal points, their composite should not exceed 18 decimal digits in length.

4. Division by 0 always causes a size-error condition if SIZE ERROR is specified. If no SIZE ERROR clause is present, division by 0 causes the program to abort.

5. The DIVIDE statement is governed by the rules for GIVING, ROUNDED, SIZE ERROR, and CORRESPONDING at the start of this chapter, and by the rules for Arithmetic Statements in Chapter 4.

6. Prime extensions: in Format 4, data-name-1 and data-name-2 must be group items. The use of arithmetic expressions in DIVIDE statements is a Prime extension.

General Rules

1. In Format 1, data-name-1 or literal-1 is divided into data-name-2, data-name-3, and so on; the quotient then replaces the dividend in data-names 2, 3, and so on.

2. In Format 2, division occurs according to these rules:

   - If the keyword INTO is used, the value of the first operand is divided into the value of the second, and the result is stored in data-name-3, data-name-4, and so on.

   - If the keyword BY is used, the value of the first operand is divided by the value of the second, and the result is stored in data-name-3, data-name-4, and so on.

3. Format 3 is used when a remainder from the division operation is desired. The remainder in COBOL is defined as the result of subtracting the product of the quotient (data-name-3) and the divisor from the dividend. If data-name-3 is defined as a numeric edited item, the quotient used to calculate the remainder is an intermediate field that contains the unedited quotient. If ROUNDED is used, the quotient used to calculate the remainder is an intermediate field that contains the quotient of the DIVIDE statement, truncated rather than rounded.

4. The accuracy of the REMAINDER data item (data-name-4) is defined by the calculation described above. Appropriate decimal alignment and truncation (not rounding) are performed for data-name-4, as needed.

5. When the ON SIZE ERROR phrase is used in Format 3, the following rules pertain:

   - If the size error occurs on the quotient, no remainder calculation is meaningful. Thus, the contents of the data items referenced by both data-name-3 and data-name-4 remains unchanged.

   - If the size error occurs on the remainder, the contents of data-name-4 remains unchanged.

6. In Format 4, corresponding elementary items subordinate to data-name-1 and data-name-2 are divided. If INTO is specified, the quotient is placed in the matching elementary items subordinate to data-name-2. Otherwise the quotient is placed in the matching elementary items subordinate to data-name-1.

EJECT -- PRIME EXTENSION

EJECT directs the compiler to start a new page for the program listing.

Format

EJECT

General Rule

This statement causes the compiler to insert a form feed in the program listing after the line containing EJECT. The statement may occur in any division in the program. It must be in coding area B (Columns 12-72).

ENTER

Function

The ENTER statement is used for documentation only. It has no effect on the compiler or the compiled program.

Format

ENTER language-name [routine-name] .

Syntax Rules

1. The language-name and routine-name following ENTER may be any user-defined word. Each must contain at least one alphabetic character.

2. A CALLed program may be written in a source language other than COBOL.

EXHIBIT

Function

The EXHIBIT statement displays data at the user terminal. It is useful for debugging.

Format

```
EXHIBIT { literal                  } ...
        { [NAMED] data-name        }
```

General Rules

1. The EXHIBIT statement may be inserted anywhere in the PROCEDURE division to provide debugging information. Specified data is exhibited on the terminal, in the formats shown for the DISPLAY statement.

2. The EXHIBIT statement differs from DISPLAY in that both the data-name and its value, connected by an = character, are displayed. The = character is preceded and followed by a space.

3. EXHIBIT is the same as EXHIBIT NAMED.

Example

Statement	Output
EXHIBIT NAMED EMPLOYEE-NO	EMPLOYEE-NO = 950

EXIT

Function

The EXIT statement provides an endpoint for a procedure or series of procedures.

Format

EXIT.

Syntax Rules

1. The EXIT statement must appear in a sentence by itself.

2. The EXIT sentence may be the only sentence in the paragraph. It should be the last sentence in its paragraph. It is never required in Prime COBOL.

EXIT PROGRAM

Function

The EXIT PROGRAM statement marks the logical end of a called program.

Format

EXIT PROGRAM.

Syntax Rules

1. The EXIT PROGRAM statement must appear in a sentence by itself.

2. The EXIT PROGRAM sentence may be the only sentence in the paragraph. If used, it should be the last sentence in its paragraph.

General Rules

1. The execution of an EXIT PROGRAM statement in a called program causes control to be returned to the calling program. An EXIT PROGRAM statement in a program that is not invoked by a CALL statement functions as an EXIT statement.

2. When the -OLD compile option is used, EXIT PROGRAM suppresses the request for interactive file assignments. (See Chapter 3.)

GO TO

Function

The GO TO statement transfers control from one part of the PROCEDURE division to another.

Format 1

GO TO [procedure-name]

Format 2

GO TO procedure-name-1 [procedure-name-2] ···, procedure-name-n

DEPENDING ON { **data-name** / **arith-expr** }

Syntax Rules

1. A paragraph or section-name referenced by an ALTER statement can consist only of that procedure-name followed by a Format-1 GO TO statement.

2. In Format 2, data-name must be an elementary numeric integer data item. The arith-expr should have an integer value.

3. A procedure-name may be either a paragraph-name or a section-name.

4. A Format-1 GO TO statement without a procedure-name must be the only statement in its paragraph.

5. The use of an arithmetic expression in a GO TO statement is a Prime extension.

General Rules

1. In Format 1, if procedure-name is not given, an ALTER statement referring to this GO TO must be executed before the GO TO is executed, otherwise control passes to the next statement.

2. When a Format-1 GO TO statement is executed, control is transferred to procedure-name, or to another procedure-name if the GO TO statement has been modified by an ALTER statement.

3. When a GO TO statement represented by Format 2 is executed, control is transferred to procedure-name-1, procedure-name-2, and so on, depending on the value of the data-name or the arithmetic expression. This value should be between 1 and n, where n is the number of procedure-names listed. If the value of the data-name or the arithmetic expression is 1, control goes to the first procedure in the series, and so on. If the value of the data-name is anything other than the positive and unsigned integers between 1 and n, then no transfer occurs and control passes to the next statement in the normal sequence for execution.

GOBACK — PRIME EXTENSION

Function

The GOBACK statement marks the logical end of a called program.

Format

GOBACK

General Rule

In a called program, execution of GOBACK returns control to the calling program. GOBACK functions in the same manner as the EXIT PROGRAM statement. See Chapter 9, INTERPROGRAM COMMUNICATION, for details.

IF

Function

The IF statement causes the evaluation of a condition, permitting the execution of specified statements depending on the value of the condition.

Format

```
IF condition ; [THEN] { NEXT SENTENCE }
                      { statement-1   }

[ ; { ELSE      } { statement-2   } ]
    { OTHERWISE } { NEXT SENTENCE }
```

Syntax Rules

1. The conditions in the IF statement must conform to the rules for conditions specified in CONDITIONAL EXPRESSIONS in Chapter 4 and in the section on Arithmetic Statements in the PROCEDURE Division at the beginning of this chapter.

2. The ELSE or OTHERWISE clauses may be omitted if they are not needed.

3. THEN and OTHERWISE are Prime extensions. THEN is always optional. OTHERWISE and ELSE are equivalent.

4. The CORRESPONDING option is used with a relation condition. Both operands of the relation must be group items.

General Rules

1. If the condition is true, either statement-1 or NEXT SENTENCE is executed as follows:

   - The statement-1, if specified, is executed. Control then passes to the next executable sentence following the IF statement, unless statement-1 contains a branch or conditional statement, in which case control is transferred according to the rules for that statement.

   - If the NEXT SENTENCE phrase is specified, control passes to the next executable sentence.

- The ELSE/OTHERWISE clause, if any, is ignored.

Examples:

```
IF BALANCE = 0 GO TO NOT-FOUND
ELSE NEXT SENTENCE.

IF X = 1.74 THEN MOVE 'M' TO FLAG OTHERWISE MOVE 'N'
     TO SECOND-TIME.


IF ACCOUNT-FIELD = SPACES OR NAME = SPACES ADD 1 TO
     SKIP-COUNT ELSE PERFORM BYPASS.
```

2. If the condition is false, statement-1 or its replacement NEXT SENTENCE is bypassed, and control passes as follows:

   - The statement-2, if specified, is executed. Control then passes to the next executable sentence, unless statement-2 contains a branch or conditional statement, in which case control is transferred according to the rules for that statement.

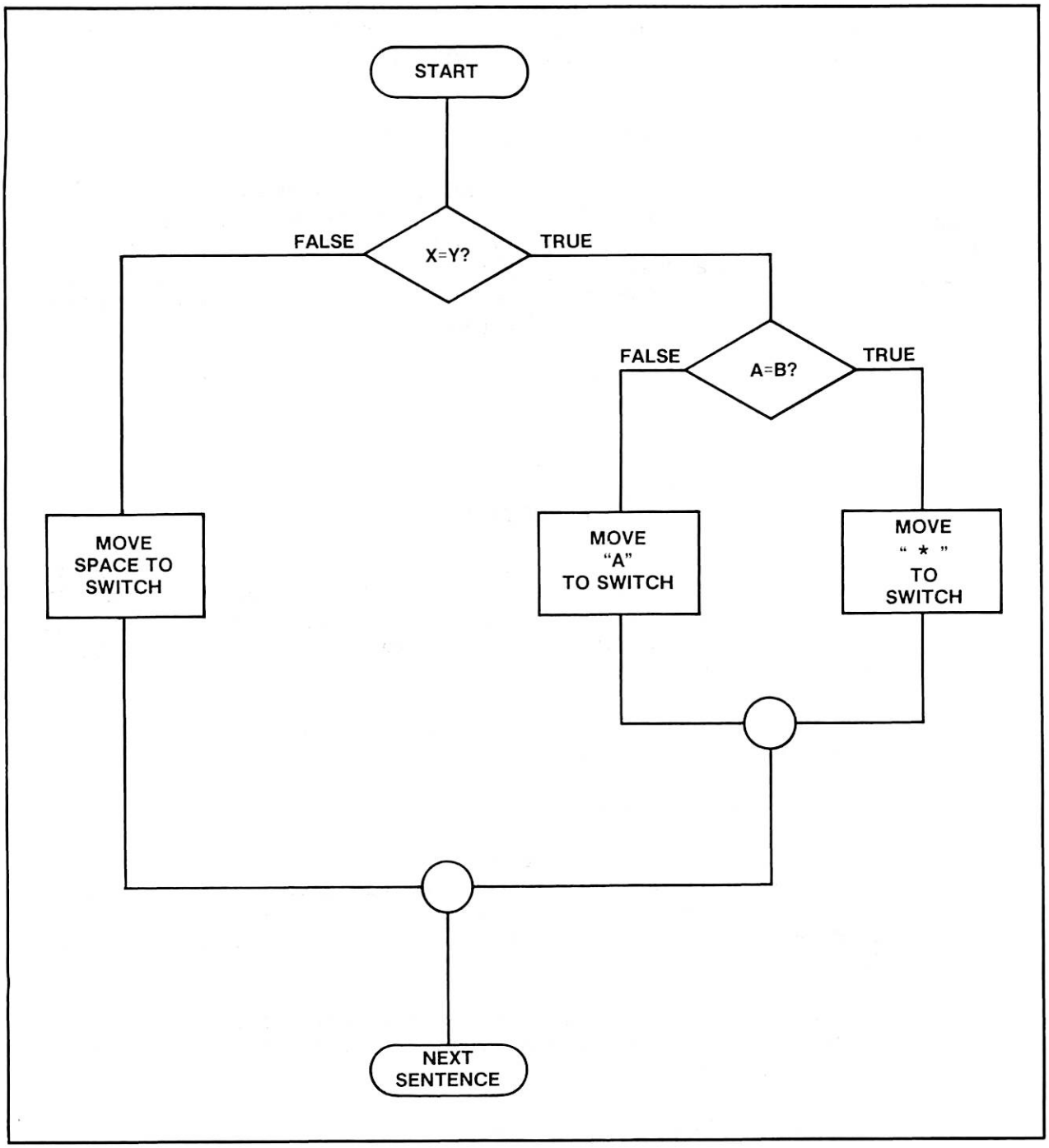
   - If no ELSE/OTHERWISE clause is specified, or if ELSE/OTHERWISE NEXT SENTENCE is specified, control passes to the next executable sentence.

3. The IF statement is said to be "nested" whenever statement-1 or statement-2 contains another IF statement. In nested IF statements, ELSEs are paired with IFs in the following way. Any ELSE encountered applies to the last preceding IF that has not been already paired with an ELSE. It is not required that the number of ELSEs in a sentence be the same as the number of IFs, but there may not be more ELSEs than IFs. OTHERWISE follows the same pairing rule as ELSE with nested IFs.

   Example:

```
IF X = Y
THEN  IF A = B
      THEN MOVE "*" TO SWITCH
      ELSE MOVE "A" TO SWITCH
ELSE MOVE SPACE TO SWITCH.
```

   The flow of this sentence may be represented by the tree structure in Figure 8-3.

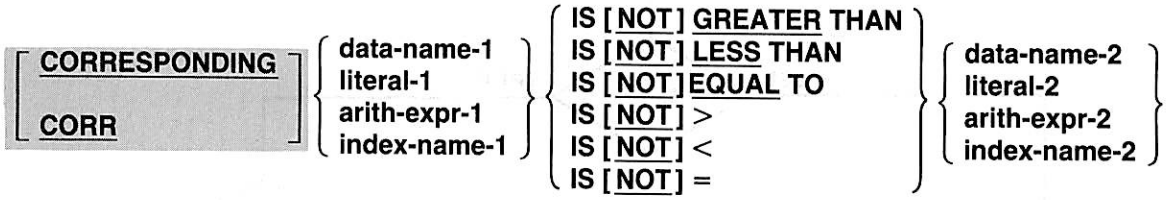


Nested IF Structure
Figure 8-3

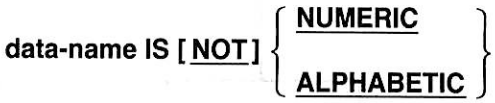
Note

The following condition types are explained in detail in Chapter 4. See Chapter 4 also for status-name, negated conditions, and combined abbreviated conditions.

4. The relation condition has the format:



5. The class condition determines whether an operand is numeric or alphabetic. Its format is:



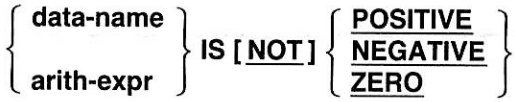
The NUMERIC test is valid only for a group, numeric DISPLAY, COMP-3, or character item. The ALPHABETIC test is valid only for a group or character item. Lowercase letters are not considered alphabetic in class tests; see Chapter 4 for a detailed discussion.

6. The condition-name condition tests the value of a conditional variable. Its format is:

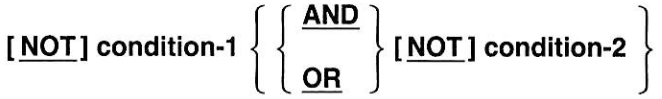
[NOT] condition-name

The condition-name is defined as a level-88 data item in a record-description-entry in the DATA division. The conditional variable is the data item immediately preceding the level-88 item or items. It may also be a switch-status name. (See Chapter 4.)

7. The sign condition tests an arithmetic expression to determine whether its value is greater than, less than, or equal to zero. The format is:



8. Two or more conditions can be combined by the logical operators AND and OR. The format for a combined condition is:



INSPECT

Function

The INSPECT statement enables the programmer to examine a character-string item and to tally, replace, or tally and/or replace occurrences of single characters or groups of characters in the item.

Format 1

```
INSPECT data-name-1 TALLYING

{ data-name-2 FOR { , { { ALL       } { data-name-3 } } }
                  {   { LEADING     } { literal-1   } }
                  { CHARACTERS                        }

[ { { BEFORE } INITIAL { data-name-4 } ] } ... } ...
  { { AFTER  }         { literal-2   } ] }
```

Format 2

```
INSPECT data-name-1 REPLACING

{ CHARACTERS BY { data-name-6 } [ { { BEFORE } INITIAL { data-name-7 } ] ]
                { literal-4   }   { { AFTER  }         { literal-5   } }

{ { , { ALL       } } { , { data-name-5 } BY { data-name-6 } }
  {   { LEADING   } }     { literal-3   }    { literal-4   }
  {   { FIRST     } }

[ { { BEFORE } INITIAL { data-name-7 } ] } ... } ... }
  { { AFTER  }         { literal-5   } ] }
```

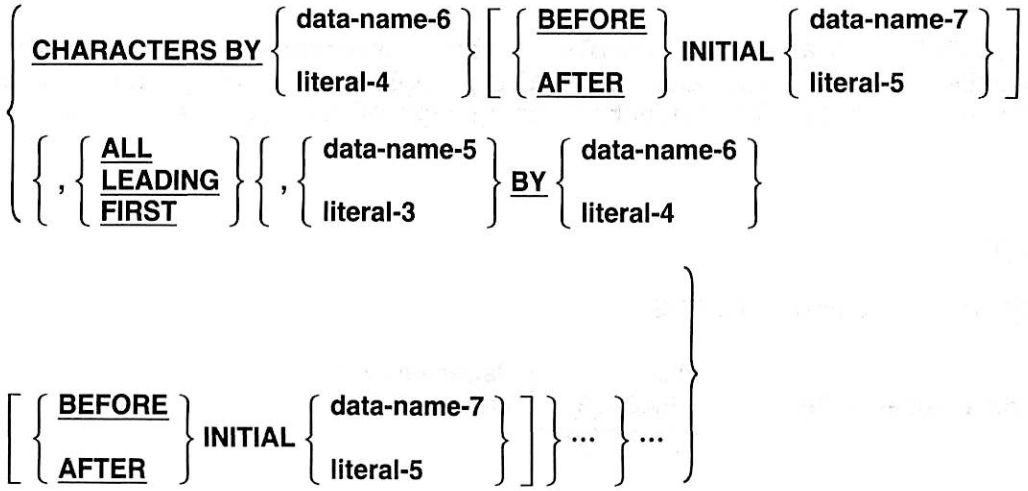
Format 3

```
INSPECT data-name-1 TALLYING

{ , data-name-2 FOR { , { { ALL       } { data-name-3 } } }
                    {   { LEADING     } { literal-1   } }
                    { CHARACTERS                        }

[ { { BEFORE } INITIAL { data-name-4 } ] } ... } ...
  { { AFTER  }         { literal-2   } ] }
```

REPLACING



Syntax Rules

1. The operand after INSPECT (data-name-1) must be a group item or an elementary item described (implicitly or explicitly) as USAGE IS DISPLAY.

2. The operands of all clauses except TALLYING may be either data items or literals. If they are data items, these operands must reference elementary alphabetic, alphanumeric or numeric items described (implicitly or explicitly) as USAGE IS DISPLAY.

3. If they are literals, each of these operands must be a nonnumeric literal and may be any figurative constant, except ALL.

4. Literals 1 through 5 and data-names 3 through 7 may be characters or groups of characters.

5. Operands of INSPECT may be no longer than 32767 bytes in length.

Rules for Formats 1 and 3

6. The operand of TALLYING (data-name-2) must be an elementary numeric data item.

7. If either literal-1 or literal-2 is a figurative constant, the figurative constant refers to an implicit one-character data item.

Rules for Formats 2 and 3

8. The size of literal-4 or data-name-6 must be equal to the size of literal-3 or data-name-5. When a figurative constant is used as literal-4, the size of the figurative constant is equal to the size of literal-3 or the size of data-name-5.

9. When the CHARACTERS phrase is used, literal-4, literal-5, or the size of data-name-6 and data-name-7 must be one character in length.

10. When a figurative constant is used as literal-3, literal-4, or data-name-6 must be one character in length.

General Rules

1. The INSPECT statement enables examination of a character-string item, permitting various combinations of the following actions:

   - Counting appearances of a specified character
   - Replacing a specified character or group of characters by an alternative
   - Qualifying and limiting the above actions according to the appearance of other specific characters

   Inspection includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying and/or replacing. It begins at the leftmost character position of data-name-1 and proceeds from left to right to the rightmost character position, as described in General Rules 4 through 6.

2. For use in the INSPECT statement, the contents of data-names 1, 3, 4, 5, 6, and 7 will be treated as follows:

   - If any of these data-names is described as alphanumeric, the INSPECT statement treats the contents of each such field as a character-string.
   - If any of these data-names is described as alphanumeric edited, numeric edited, or unsigned numeric, the data item is inspected as though it had been redefined as alphanumeric and the INSPECT statement referred to the redefined data item.
   - If any of these data-names is described as signed numeric, the data item is inspected as though it had been moved to an unsigned numeric data item of the same length.

3. All references to literal-1, literal-2, literal-3, literal-4, and literal-5 apply equally to the contents of data-names 3, 4, 5, 6, and 7, respectively.

4. During inspection of the contents of data-name-1, each properly matched occurrence of literal-1 is tallied (Formats 1 and 3) and/or each properly matched occurrence of literal-3 is replaced by literal-4 (Formats 2 and 3).

5. The comparison operation to determine the occurrences of literals to be tallied and/or replaced (literals 1 and 3) occurs as follows:

   - The operands of the TALLYING and REPLACING phrases are considered in the order they are specified in the INSPECT statement. The first literal is compared to an equal number of contiguous characters, starting with the leftmost character position in the data item referenced by data-name-1. The literal and that portion of data-name-1 match if, and only if, they are equal character for character.

   - If no match occurs in the comparison of the first literal, the comparison is repeated with each successive literal, if any, until either a match is found or there is no next successive literal. When there is no next successive literal, the character position in data-name-1 immediately to the right of the leftmost character position considered in the last comparison cycle is considered as the leftmost character position, and the comparison cycle begins again with the first literal.

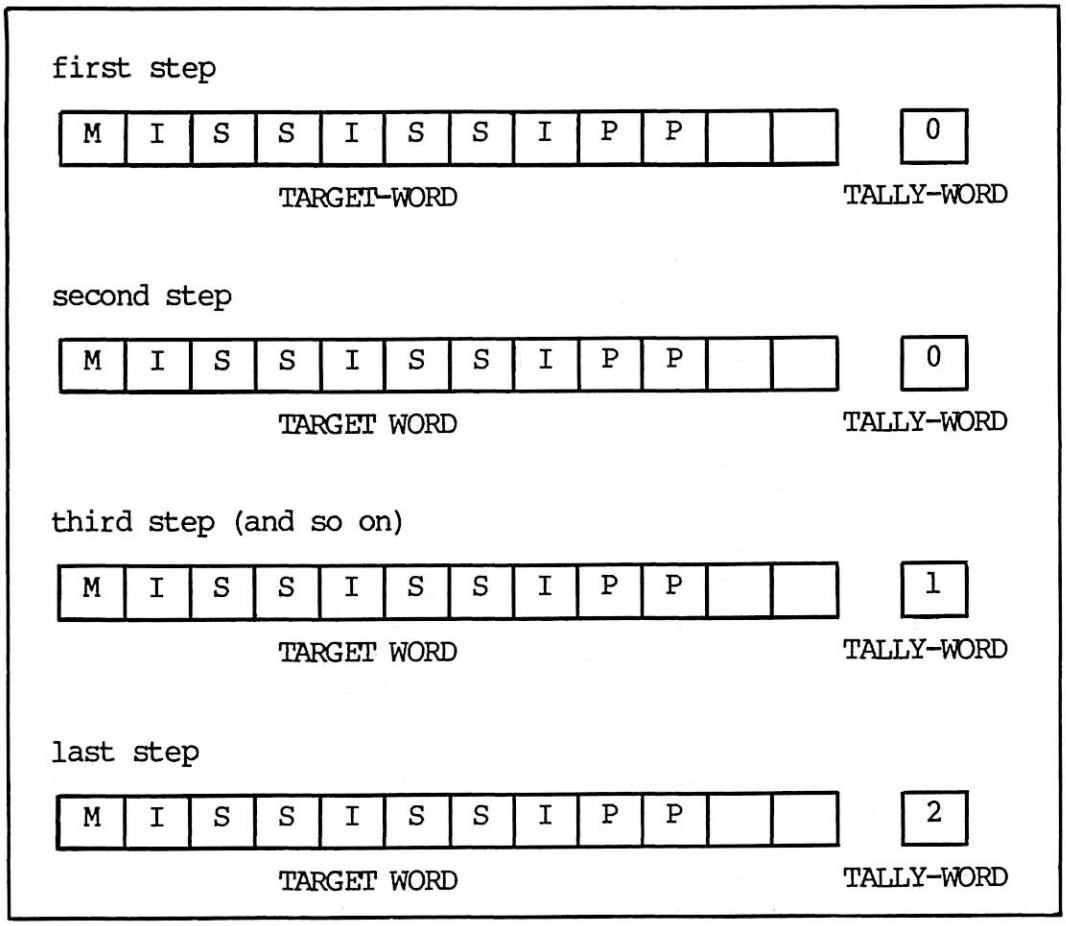
   - Whenever a match occurs, tallying and/or replacing takes place as described in General Rules 8 and 9. The character position in data-name-1 immediately to the right of the rightmost character position that participated in the match is now considered to be the leftmost character position of data-name-1 and the comparison cycle starts again with the first literal.

   - The comparison operation continues until the rightmost character position of data-name-1 has participated in a comparison or has been considered as the leftmost character position. When this occurs, inspection is terminated.

This series of steps may be represented as in Figure 8-4, for the statements

```
MOVE 0 TO TALLY-WORD.
INSPECT TARGET-WORD TALLYING TALLY-WORD FOR ALL 'SS'.
```

The solid lines mark the two characters being inspected for a match with 'SS' in each step.



Steps in INSPECT ... TALLYING
Figure 8-4

- If the CHARACTERS phrase is specified, an implied one-character operand participates in the cycle described in the preceding four paragraphs, except that no comparison to the contents of data-name-1 takes place. This implied character is considered always to match the leftmost character of data-name-1 participating in the current comparison cycle.

6. If the BEFORE or AFTER phrase is not specified, literal-1, literal-3, or the implied operand of the CHARACTERS phrase participates in the comparison operation.

If the BEFORE phrase is specified, the associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase participates only in those comparison cycles which involve that portion of data-name-1 from its leftmost character position up to, but not including, the first occurrence of literal-2, literal-5. The position of this first occurrence is determined before the first cycle of the comparison operation is begun. If, on any comparison cycle, literal-1, literal-3, or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of data-name-1. If there is no occurrence of literal-2, literal-5 within data-name-1, then the associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase participates in the comparison operation as though the BEFORE phrase had not been specified.

If the AFTER phrase is specified, the associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase may participate only in those comparison cycles which involve that portion of data-name-1 from the character position immediately to the right of the rightmost character position of the first occurrence of literal-2, literal-5. The position of this first occurrence is determined before the first cycle of the comparison operation is begun. If, on any comparison cycle, literal-1, literal-3, or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of data-name-1. If there is no occurrence of literal-2, literal-5 within data-name-1, then the associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase is never eligible to participate in the comparison operation.

7. The content of data-name-2 is not initialized by execution of the INSPECT statement.

8. The TALLYING clause causes character-by-character or character-group by character-group comparison, from left to right, of data-name-1 with data-name-3 or literal-1. The count is accumulated in data-name-2. See example 1 below.

   - When the AFTER INITIAL clause is present, the counting process begins only after detection of a character or character-group in data-name-1 matching the operand following INITIAL. If BEFORE INITIAL operand is specified, the counting process terminates upon encountering a character in data-name-1 that matches the operand following INITIAL. See examples 2 and 4 below.

   - If the ALL phrase is specified, the content of data-name-2 is incremented by one for each occurrence of the operand after FOR matched within the content of data-name-1.

- If the LEADING phrase is specified, the content of data-name-2 is incremented by one for each contiguous occurrence of the operand after FOR matched within the content of data-name-1, provided that the leftmost such occurrence is at the point where the comparison began and where the operand after FOR was eligible to participate.

- If the CHARACTERS phrase is specified, the content of data-name-2 is incremented by one for each character in data-name-1.

9. The reserved words ALL, LEADING, and FIRST apply to each succeeding BY phrase until the next adjective appears.

10. The REPLACING clause causes replacement of characters under specified conditions.

    - If BEFORE INITIAL operand is present, replacement does not continue after detection of a character in data-name-1 matching the operand after INITIAL. If AFTER INITIAL is present, replacement does not commence until detection of a character in data-name-1 matching the operand after INITIAL.

    - If the ALL phrase is specified, each occurrence of the operand after REPLACING matched within the content of data-name-1 is replaced by the operand after BY.

    - If the LEADING phrase is specified, each contiguous occurrence of the operand after REPLACING matched within the content of data-name-1 is replaced by the operand after BY, provided that the leftmost occurrence is at the point where the comparison began and where the operand after REPLACING was eligible to participate. See example 6 below.

    - If the FIRST phrase is specified, the leftmost occurrence of the operand after REPLACING matched within the content of data-name-1 is replaced by the operand after BY. See example 5 below.

    - When the CHARACTERS phrase is specified, each character in data-name-1 is replaced by the operand after BY. See example 3 below.

11. A Format-3 INSPECT statement is executed as though two separate INSPECT statements were written. The first contains only a TALLYING clause, the second contains only a REPLACING clause. See example 4 below.

Examples

1. INSPECT name TALLYING counter FOR ALL 'L'.

Name Before	Counter After	Name After
LILLY	3	LILLY
SMALL	2	SMALL

2. INSPECT name TALLYING counter FOR LEADING 'B'
AFTER INITIAL 'A'
REPLACING CHARACTERS BY 'X'.

Name Before	Counter After	Name After
ABACK	1	XXXXX
CABBAGE	2	XXXXXXX

3. INSPECT name REPLACING CHARACTERS BY '$'
BEFORE INITIAL '.'.

Name Before	Counter After	Name After
AB D.99		$$$$$.99

4. INSPECT name TALLYING counter FOR CHARACTERS
AFTER INITIAL 'E'
REPLACING ALL 'B' BY 'A'.

Name Before	Counter After	Name After
DEBATE	4	DEAATE
IBEX	1	IAEX

5. INSPECT name REPLACING FIRST 'A' BY 'P'
AFTER INITIAL 'M'.

Name Before	Counter After	Name After
LLAMAA		LLAMPA
LLOYD		LLOYD

6. INSPECT "ABC/DEF" REPLACING LEADING "ABC" BY
"123".

Name Before	Counter After	Name After
ABC/DEF		123/DEF

MERGE

Function

Combines two or more sorted files on a set of specified keys, and during the process makes records available, in merge order, to an output procedure or file.

Format

```
MERGE file-name-1 ON { ASCENDING  } KEY data-name-1 [, data-name-2] ...
                     { DESCENDING }

    [ ON { ASCENDING  } KEY data-name-3 [, data-name-4] ... ] ...
         { DESCENDING }

[COLLATING SEQUENCE IS alphabet-name]

USING file-name-2, file-name-3 [, file-name-4] ...

{ OUTPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ] }
{                                     { THRU    }                 }
{ GIVING file-name-5                                              }
```

The MERGE statement is discussed in Chapter 11.

MOVE

Function

The MOVE statement transfers data from one area of main storage to another, performing conversion and editing as indicated.

Format 1

```
MOVE { data-name-1 }
     { literal     } TO data-name-2 [ROUNDED] [, data-name-3 [ROUNDED]] ...
     { arith-expr-1}

    [, ON SIZE ERROR imperative-statement]
```

Format 2

```
MOVE { CORRESPONDING } data-name-1 TO data-name-2 [ROUNDED]
     { CORR          }

    [, ON SIZE ERROR imperative-statement]
```

Syntax Rule

1. The data-name-1, arith-expr, and the literal represent the sending area; data-name-2, data-name-3 represent the receiving area.
2. When the CORRESPONDING clause is used, both operands must be group items.
3. An index data item cannot be an operand of MOVE.
4. The use of ON SIZE ERROR, ROUNDED, and of an arithmetic expression in a MOVE statement is a Prime extension.

General Rules

1. When a group item is a receiving field, characters are moved without conversion and without editing.

2. During elementary moves editing occurs and alignment is performed according to the alignment rules in the section DATA REPRESENTATION AND ALIGNMENT in Chapter 4. Any necessary data conversion is done as described in the following rules. Legality of moves is summarized in Table 8-2.

3. Moves of numeric items to numeric or numeric edited fields follow these rules:

   - The items are aligned by decimal point, with generation of zeros or truncation on either end, as required.
   - When the types of the source field and receiving field differ, conversion to the type of the receiving field takes place.
   - If the receiving field is numeric edited, the item moved may have special editing performed on it such as suppression of zeros, insertion of a dollar sign, and decimal point alignment, as specified by the PICTURE clause of the receiving field.
   - Conversion of signs follows these rules. When the receiving item is signed, the sign of the sending item is placed in it. If the sending item is unsigned, a positive sign is generated. When the receiving item is unsigned, the absolute value of the sending item is moved.

4. When the sending item is alphanumeric and the receiving area is numeric or numeric edited, data is moved as if the sending item were an unsigned integer. The discussion in General Rule 3 applies.

5. For moves to and from alphabetic and alphanumeric fields, the following rules apply:

   - The characters are placed in the receiving area from left to right (unless JUSTIFIED RIGHT applies).
   - If the receiving field is not completely filled by data being moved, the remaining positions are filled with spaces.
   - If the source field is longer than the receiving field, the move is terminated as soon as the receiving field is filled, resulting in right truncation.

6. If the sending item is signed numeric and the receiving field is alphanumeric, the sign is not moved. If the sign occupies a separate character position, that character is not moved. The discussion in General Rule 5 applies.

7. If the sending item is nonnumeric and the receiving item is numeric, the sending field is assumed to be a numeric integer, and a numeric move is generated.

8. When overlapping fields are used as operands, results are undefined.

9. When the CORRESPONDING option is used, data-name-1 and data-name-2 must be group items. Elementary items subordinate to data-name-1 are moved to matching items subordinate to data-name-2. The move is treated as a series of elementary moves. See the rules for CORRESPONDING at the beginning of Chapter 8.

10. The SIZE ERROR and ROUNDED options can be specified only for numeric MOVEs. The rules are the same as those given for arithmetic statements at the beginning of this chapter.

11. Table 8-2 summarizes the various types of moves permitted with the MOVE statement.

Table 8-2
Moves by Category

(Y = Permitted)

Sending Data Item \ Receiving Field	Alphabetic	Alphanumeric Edited	Numeric Integer	Numeric Noninteger	Numeric Edited	Alphanumeric	Comp	Comp-1	Comp-2	Comp-3
Alphabetic	Y	Y					Y			
Alphanumeric edited	Y	Y					Y			
Numeric integer		Y	Y	Y	Y	Y	Y	Y	Y	Y
Numeric noninteger		Y	Y	Y	Y	Y	Y	Y	Y	Y
Numeric edited	Y	Y	Y	Y			Y	Y	Y	Y
Alphanumeric	Y	Y	Y(1)	Y(1)	(2)		Y	(2)	(2)	(2)	Y(1)
COMP		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
COMP-1		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
COMP-2		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
COMP-3		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

(1) Permitted. Sending field must contain only digits 0-9, +, or -.

(2) Permitted. Sending field must contain only digits 0-9.

MULTIPLY

Function

The MULTIPLY statement computes the product of two numeric data items and stores the result.

Format 1

```
MULTIPLY { data-name-1 | literal-1 | arith-expr-1 } BY data-name-2 [ROUNDED]

    [, data-name-3 [ROUNDED]] ...

    [; ON SIZE ERROR imperative-statement]
```

Format 2

```
MULTIPLY { data-name-1 | literal-2 | arith-expr-2 } BY { data-name-2 | literal-3 | arith-expr-3 }

    GIVING data-name-3 [ROUNDED] [, data-name-4 [ROUNDED]] ...

    [; ON SIZE ERROR imperative-statement]
```

Format 3

```
MULTIPLY { CORRESPONDING | CORR } data-name-1 BY data-name-2 [ROUNDED]

    [; ON SIZE ERROR imperative-statement]
```

Syntax Rules

1. Each data-name must refer to an elementary numeric item, except that in Format 2 the operands after GIVING must be elementary numeric or numeric edited.

2. Each literal must be a numeric literal.

3. The maximum size of each operand is 18 decimal digits. The composite of operands in Format 1 must not contain more than 18 decimal digits.

4. In Format 3, data-name-1 and data-name-2 must be group items.

5. The use of CORRESPONDING and of arithmetic expressions are Prime extensions.

General Rules

1. In Format 1 the product will be stored in data-name-2.

2. When the GIVING option is used, the product is stored in data-name-3, data-name-4, etc.

3. The rules for signs are those presented in ALGEBRAIC SIGNS in Chapter 4.

4. The MULTIPLY statement is governed by the rules for GIVING, ROUNDED, and SIZE ERROR in the rules for Arithmetic Statements in the PROCEDURE Division at the start of this chapter, and by the rules for Arithmetic Statements in Chapter 4.

5. In Format 3, elementary items subordinate to data-name-1 are multiplied by matching elementary items subordinate to data-name-2. Each result is stored in the associated elementary item subordinate to data-name-2.

6. The rules for CORRESPONDING are given at the beginning of Chapter 8.

NOTE -- PRIME EXTENSION

Function

NOTE allows comment entries in the PROCEDURE division.

Format

NOTE comment-entry.

Syntax Rules

1. NOTE may be followed by an entry of any length.

2. If NOTE is the first statement in a paragraph, the entire paragraph is treated as a comment by the compiler. If NOTE is not the first statement in a paragraph, only the text through the first period is treated as a comment.

General Rule

NOTE may appear only in the PROCEDURE division.

Example

```
MOVE ALL 1 TO ITEM-1.
NOTE  THE FOLLOWING TEST ALLOWS OPENING OF A SECOND DISK INPUT FILE
      IF THE SERIAL NUMBER IS ALL 1's, OTHERWISE  DISK-FILE-1  WILL
      BE CLOSED, REOPENED, AND READ AGAIN.
IF NAME =  ITEM-1  PERFORM  150-SECOND-INPUT  OTHERWISE  PERFORM
      180-REOPEN.
```

OPEN

Function

The OPEN statement initiates the processing of files and enables other input/output operations, such as label checking and writing.

Format 1

```
OPEN { INPUT    file-name-1 [, file-name-2] ... }
     { OUTPUT   file-name-3 [, file-name-4] ... }
     { I-O      file-name-5 [, file-name-6] ... } ...
     { EXTEND   file-name-7 [, file-name-8] ... }
```

Format 2

```
OPEN { INPUT    file-name-1 [, file-name-2] ... }
     { OUTPUT   file-name-3 [, file-name-4] ... }
     { I-O      file-name-5 [, file-name-6] ... } ...
     { EXTEND   file-name-7 [, file-name-8] ... }
```

Syntax Rules

1. For each file, an OPEN statement must be executed prior to a READ, WRITE, DELETE, START, CLOSE, or REWRITE statement for that file.

2. The files referred to in the OPEN statement need not all have the same organization or access.

3. The EXTEND phrase can be used only for sequential files.

Note

The information here pertains to sequential files. See Chapters 12 and 13 for information on indexed files and relative files, respectively.

General Rules

1. Format 1 is used for sequential I-O.

2. Format 2 is used for indexed I-O and relative I-O.

3. A file opened as INPUT can only be accessed in a READ statement.

4. A file opened as OUTPUT can only be accessed in a WRITE statement.

5. A file opened as I-O can be accessed by a READ, REWRITE (disk only), nonsequential DELETE, or WRITE statement.

6. A file may be opened with the INPUT, OUTPUT, EXTEND, and I-O phrases in the same program. Following the initial OPEN, each subsequent OPEN statement for that same file must be preceded by a CLOSE statement for that file.

7. For a sequentially organized file, OPEN OUTPUT file-name causes PRIMOS to create a file, writing standard labels if they are specified in the corresponding file-description-entry. OPEN with the EXTEND or INPUT option requires that the file designated by file-name already exist.

8. The file-description-entry for files with INPUT, I-O, or EXTEND (file-name-1, file-name-2, file-name-5, file-name-6, file-name-7, or file-name-8) must be equivalent to that used when this file was created. These are files that already exist.

9. The current record pointer is a conceptual flag pointing to the next record to be accessed. For files being opened with the INPUT or I-O phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. If no records exist in the file, the current record pointer is set such that the next executed READ statement for the file will result in an AT END condition.

10. When the EXTEND phrase is specified, the OPEN statement opens the file, then positions to the bottom of that file (immediately following the last logical record). Subsequent WRITE statements to the file will add records as though the file had been opened with the OUTPUT phrase, with the current record pointer positioned at end of file.

11. When the EXTEND phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

    - The beginning file labels are processed.
    - The existing ending file labels are processed as though the file is being opened with the INPUT phrase. These labels are then deleted.
    - Processing then proceeds as though the file had been opened with the OUTPUT phrase.

12. No statement that references a given file can be executed, except for the SORT and MERGE statements, until an OPEN statement is successfully executed for that file.

13. If the desired file cannot be opened, then the program will terminate abnormally at execution time with the error message FILE NOT FOUND.

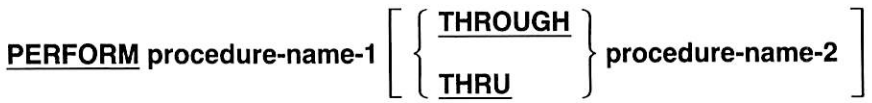
14. A file referenced in a SORT or MERGE statement may not be open at the time of execution of the SORT or MERGE statement. Once a file has been opened under control of SORT or MERGE, no I-O operation other than those under control of SORT or MERGE may be executed until the sort or merge is completed.

PERFORM

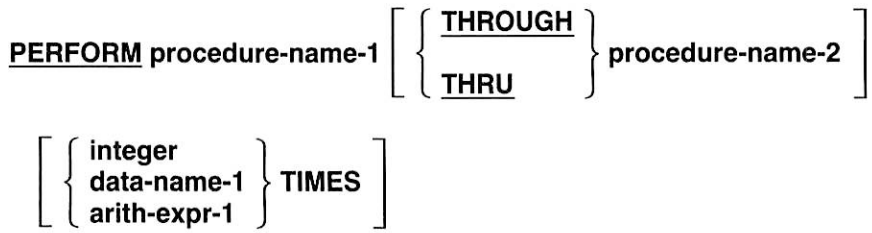
Function

The PERFORM statement is used to transfer control explicitly to one or more procedures, and to return control implicitly to the normal sequence after execution of the specified procedures.

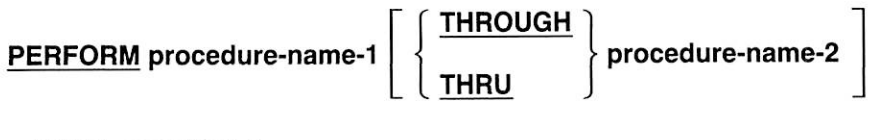
Format 1



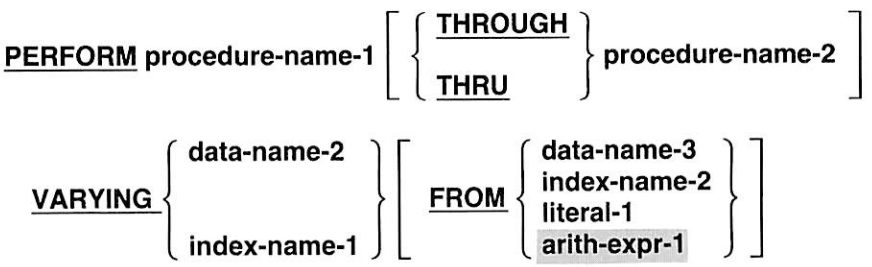
Format 2

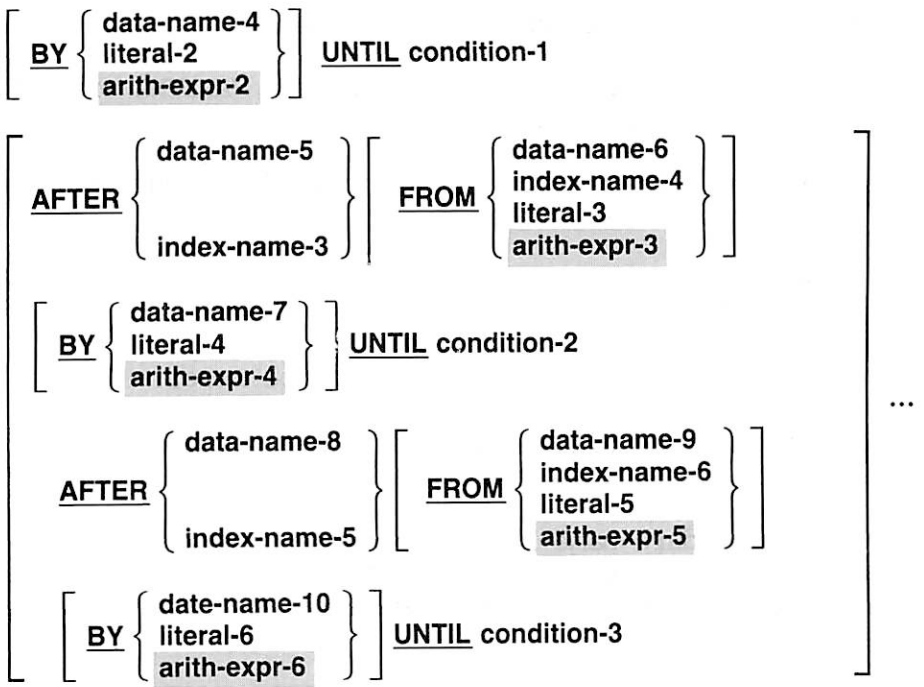


Format 3



Format 4





Syntax Rules

1. The words THROUGH and THRU are equivalent.

2. Each data-name represents an elementary numeric item described in the DATA division. In Format 2, data-name-1 must represent a numeric integer.

3. Each literal represents a numeric integer.

4. A procedure-name may be either a section-name or a paragraph-name.

5. If an index-name is used in the VARYING or AFTER phrase, then:

   - The data-name in the associated FROM and BY phrases must be an integer data item.
   - The literal in the associated FROM phrase must be a positive integer. The arithmetic expression must evaluate to a positive integer.
   - The literal in the associated BY phrase must be a nonzero integer. The arithmetic expression must evaluate to a positive integer.

6. If an index-name is specified in the FROM phrase, then:

   - The data-name in the associated VARYING or AFTER phrase must be an integer data item.
   - The data-name in the associated BY phrase must be an integer data item.
   - The literal in the associated BY phrase must be an integer.

7. A literal in the BY phrase must not be zero.

8. Condition-1, condition-2, condition-3 may be any conditional expression described in Chapter 4.

9. Where procedure-name-1 and procedure-name-2 are both specified and either is the name of a procedure in a declarative section, then both must be paragraph-names in the same declarative section.

10. The use of arithmetic expressions in FROM and BY clauses is a Prime extension.

11. If the FROM and BY clauses are omitted, the value 1 is assumed.

General Rules

1. The data-names 4, 7, and 10 must not have a zero value.

2. If the PERFORM statement is written with no options, control is transferred to the first statement of procedure-name-1. At the completion of procedure-name-1, control is returned to the next executable statement following the PERFORM statement.

3. If procedure-name-2 is specified and it is a paragraph-name, then control is returned to the statement following the PERFORM after the last statement of that paragraph is executed.

4. If procedure-name-2 is specified and it is a section-name, then control is returned to the statement following the PERFORM after the last statement of the last paragraph of that section is executed.

5. In Formats 1 and 2:

   If the THROUGH option is used, multiple paragraphs or sections can be executed before control is returned to the statement after PERFORM.

In Format 2:

If the TIMES option is used, procedures are performed the number of times meant by data-name-1, arithmetic-expression-1, or integer.

If data-name-1, arithmetic-expression-1, or integer is initially zero or negative, the PERFORM statement is not executed; control passes to the statement following the PERFORM statement.

During execution of the PERFORM statement, if the value of data-name-1 or arithmetic-expression-1 is changed, the number of times the procedure is executed will nevertheless be that of the initial value of data-name-1.

6. In Formats 3 and 4, the condition may be any type of condition, including the CORRESPONDING relation condition.

7. In Format 3:

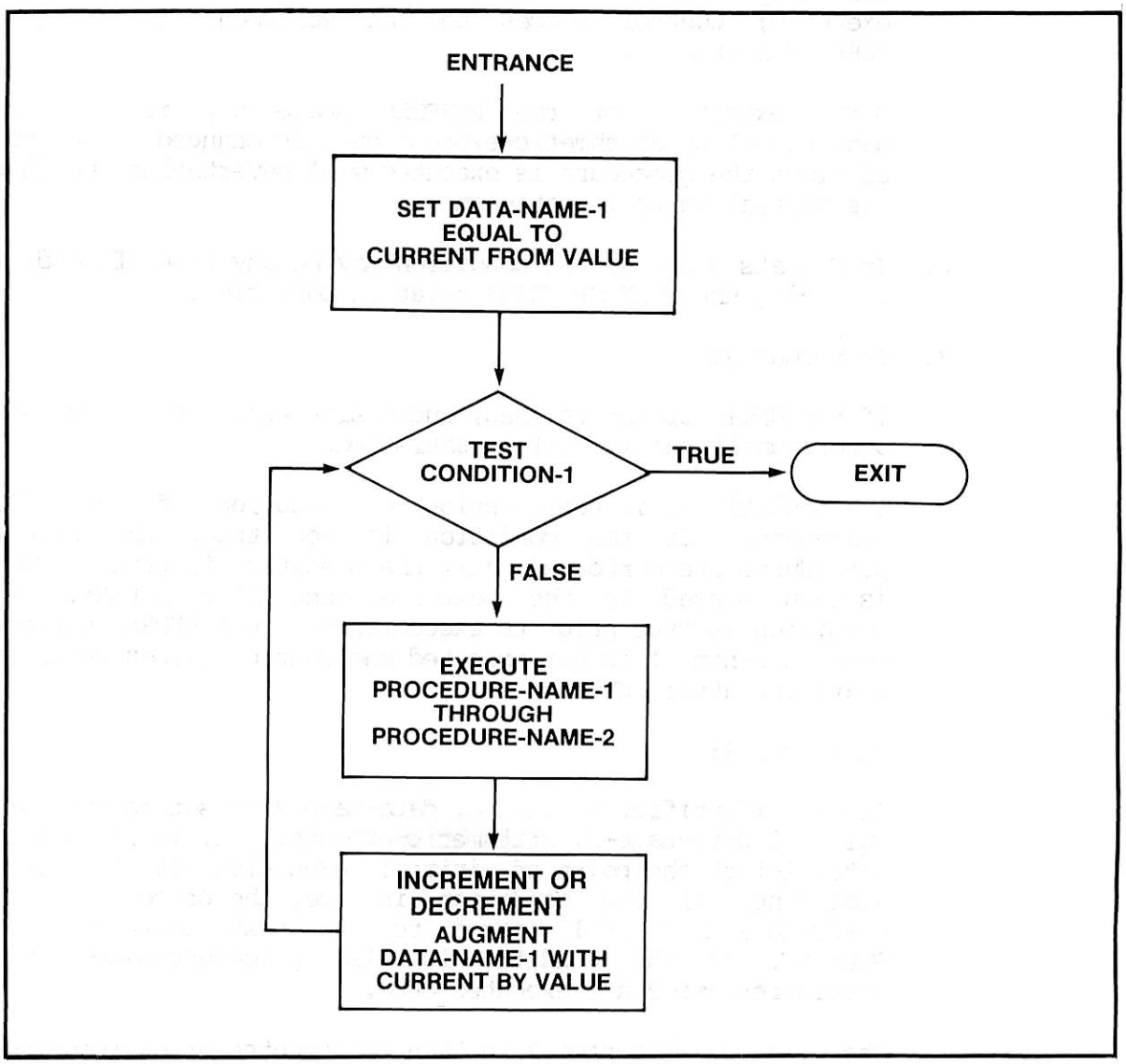
   If the UNTIL option is used, successive execution of procedures occurs until condition-1 is satisfied.

   The condition-1 is tested prior to execution of the PERFORM statement. If the condition is not true, the specified procedures are performed until the condition is true. Control is then passed to the next statement after PERFORM. If the condition is true prior to execution of the PERFORM statement, procedure-name-1 is not executed and control passes to the next statement after PERFORM.

   In Format 4:

   If one identifier is varied, data-name-2 is set to the current value of data-name-3, arithmetic-expression-1, index-name-1, or literal-1 at the point of initial execution of the PERFORM statement. If the condition is true, the procedures are not executed and control passes to the next statement after PERFORM. If the condition is false, procedure-name-1 through procedure-name-2 are executed once.

   The value of data-name-2 is then incremented or decremented by the value in data-name-4, arithmetic-expression-2, or literal-2. The condition is reevaluated. The cycle continues until the condition is true, at which point control is transferred to the next statement following the PERFORM statement. See Figure 8-5.



Logic of PERFORM Statement (One Identifier Varied)
Figure 8-5

At the termination of the PERFORM statement, data-name-2 or index-name-1 has a value that differs from the last used setting by the value of data-name-4, arithmetic-expression-2, or literal-2. If the condition was true before initial execution of the PERFORM statement, data-name-2 or index-name-1 contains the current value of data-name-3, literal-1, arithmetic-expression-1, or index-name-2.

When two identifiers are varied, data-name-2 and data-name-5 are set to the current value of data-name-3 and data-name-6, respectively. Condition-1 is then evaluated. If it is true, control is transferred to the next statement; if false, condition-2 is evaluated. If condition-2 is false, procedure-name-1 through procedure-name-2 are executed once, then data-name-5 is augmented by data-name-7, arithmetic-expression-4, or literal-4, and condition-2 is evaluated again. This cycle of evaluation and augmentation continues until condition-2 is true. When condition-2 is true, data-name-5 or index-name-3 is set to the value of literal-3, data-name-6, index-name-4 (if index-name-3 is being varied), or arithmetic-expression-3.

Data-name-2 is augmented by data-name-4, literal-2, or arithmetic-expression-2, and condition-1 is reevaluated. The PERFORM statement is completed if condition-1 is true; if not, the cycles continue until condition-1 is true.

During the execution of the procedures associated with the PERFORM statement, any change to the VARYING data-name, the BY data-name, the AFTER data-name, or the FROM data-name will be taken into consideration and will affect the operation of the PERFORM statement.

In Format 4, data-name-5 goes through a complete cycle (FROM, BY, UNTIL) each time data-name-2 is varied. See Figure 8-6.

At the termination of the PERFORM statement, data-name-5 contains the current value of data-name-6. Data-name-2 has a value that exceeds the last used setting by one increment or decrement value, unless condition-1 was true when the PERFORM statement was entered, in which case data-name-2 contains the current value of data-name-3.

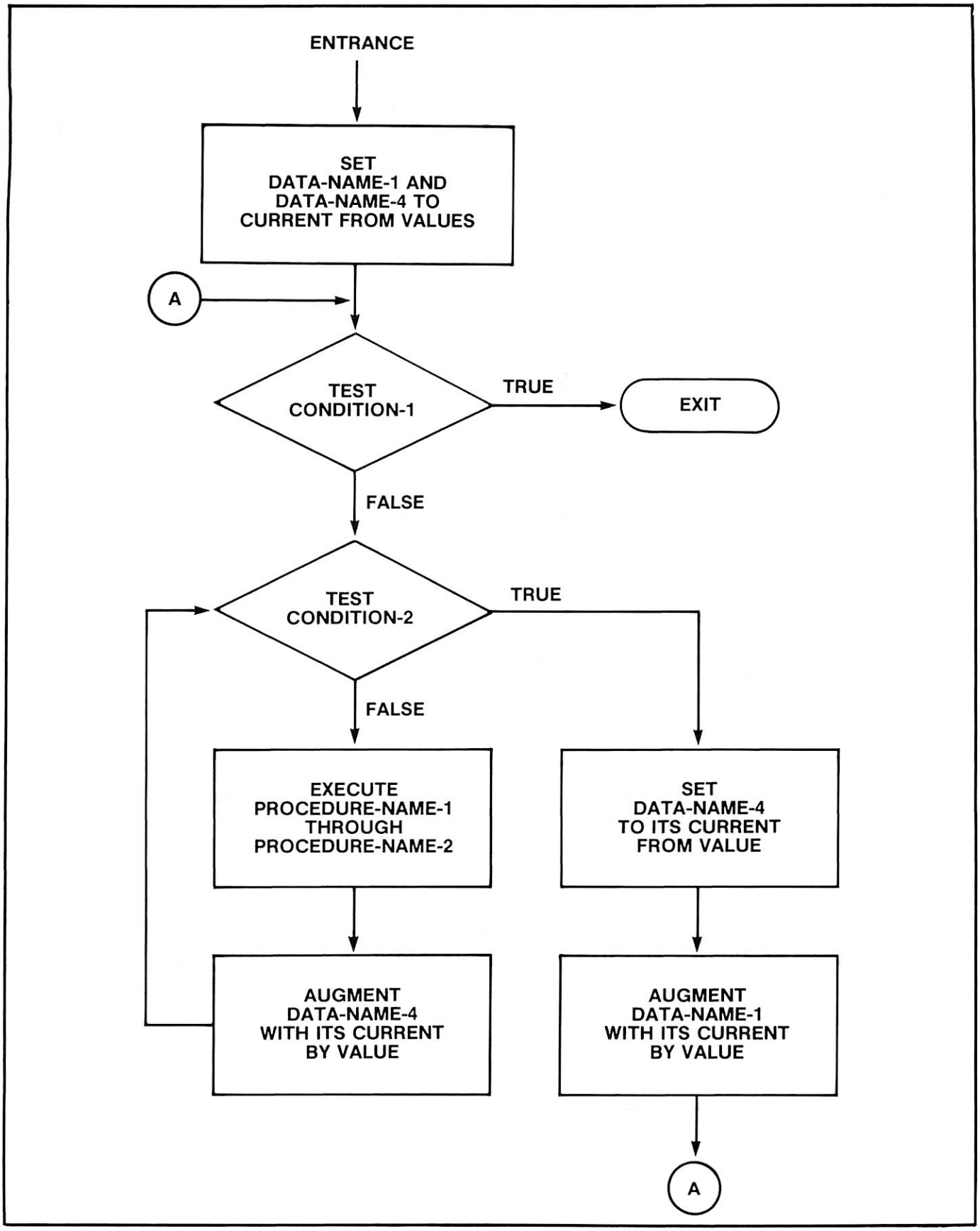
When three or more identifiers are varied, the mechanism is an extension of the one for two identifiers. See Figure 8-7.

After the completion of the PERFORM statement, each data item varied by an AFTER phrase contains the current value of the data-name in the associated FROM phrase. Data-name-2 has a value that exceeds its last used setting by one increment or decrement value, unless condition-1 is true when the PERFORM statement is entered, in which case data-name-2 contains the current value of data-name-3.

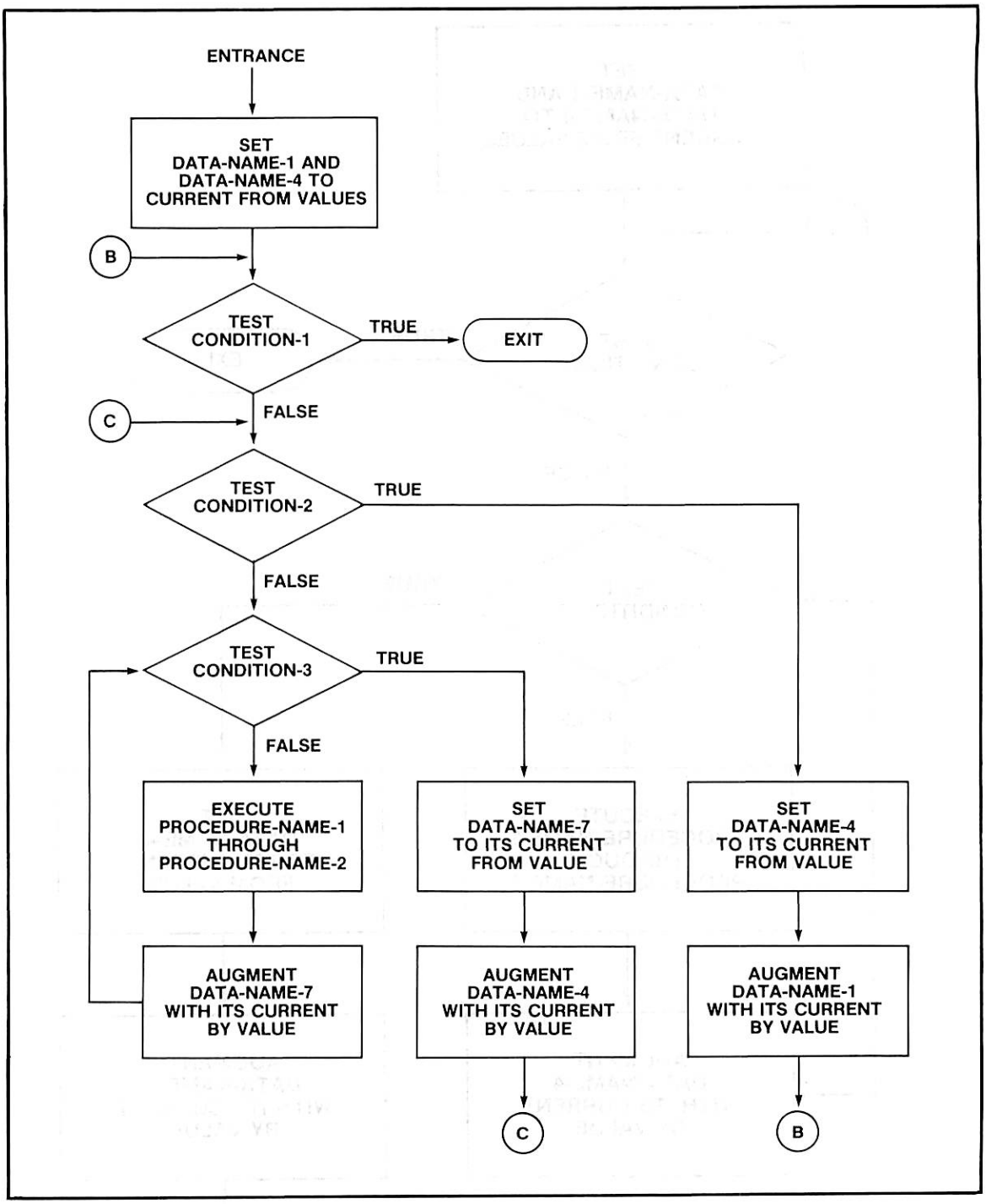
An example for a Format-4 PERFORM statement is shown below:

```
                    .
                    .
                    .
START-PARA.
    PERFORM INT-PARA
        VARYING INDX1 FROM 1 BY 1
          UNTIL INDX1 > 2
        AFTER INDX2 FROM 1 BY 1
          UNTIL INDX2 > 12
        AFTER INDX3 FROM 1 BY 1
          UNTIL INDX3 > 7.
INT-PARA.
        MOVE ZEROS TO DEPT-TOTAL(INDX1, INDX2, INDX3).
                    .
                    .
                    .
```

8. If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence encompassed by the first PERFORM. Thus, an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement, must not allow control to pass to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements may not have a common exit. See Figure 8-8.



Logic of PERFORM Statement (Two Identifiers Varied)
Figure 8-6



Logic of PERFORM Statement (Three Identifiers Varied)
Figure 8-7

READ

Function

The READ statement makes available a record from a file.

Format 1

READ file-name RECORD [INTO data-name-1]

[; AT END imperative-statement]

Format 2

READ file-name [NEXT] RECORD [INTO data-name-1]

[; AT END imperative-statement]

Format 3

READ file-name RECORD [INTO data-name-1]

[; INVALID KEY imperative-statement]

Format 4

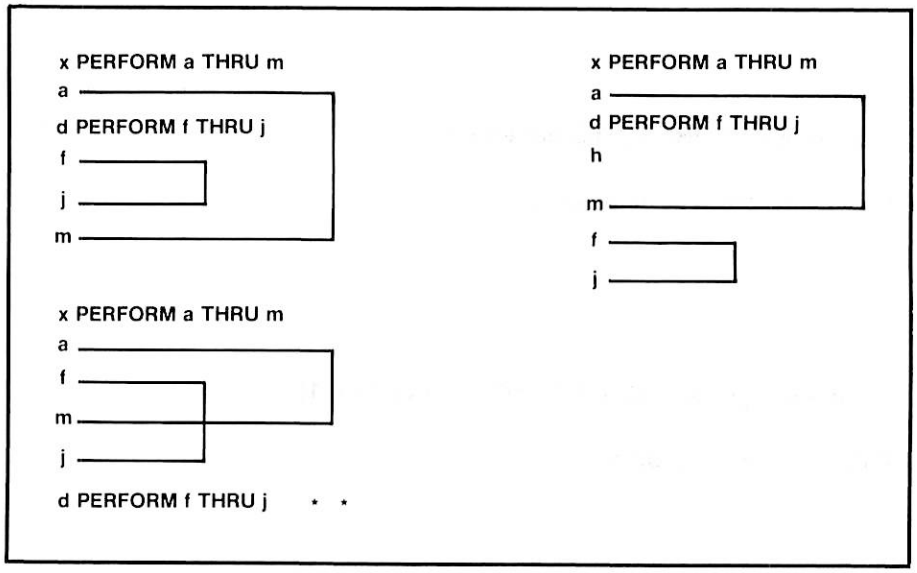
READ file-name RECORD [INTO data-name-1]

[; KEY IS data-name]

[; INVALID KEY imperative-statement]

Syntax Rules

1. Format 1 is used for all sequentially organized files.
2. The NEXT phrase option in Format 2 is used only for indexed and relative files, in SEQUENTIAL or DYNAMIC access modes, when records are to be retrieved sequentially.
3. Formats 3 and 4 are used only for indexed and relative files.
4. The KEY IS option is used only for indexed files.



**Note that, in the last example, statement (d) must not be executed while the first PERFORM statement (x) is active, or program-control code at m would cause an erroneous return to the statement following x.

Permissible PERFORM Sequences
Figure 8-8

5. The AT END phrase must be specified if no applicable USE procedure is specified for file-name.

Note

This discussion of the READ statement includes sequentially-organized files only. Further detailed discussion of READ statement formats as they apply to indexed files and relative files will be found in Chapters 12 and 13, respectively.

General Rules for READ with Sequential Files

1. A file must be OPEN for INPUT, EXTEND, or I-O when a READ statement for that file is executed.

2. Execution of the READ statement makes a record available to the program before execution of any subsequent statement, provided AT END is not invoked. To do so, it makes use of the current record pointer, a conceptual entity that points to the next record to be accessed. The record to be made available by the READ statement is determined as follows:

   - If the current record pointer was positioned by the execution of the OPEN statement, the record pointed to by the current record pointer is made available.

   - If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file and then that record is made available.

3. The execution of the READ statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. See FILE STATUS under FILE-CONTROL in Chapter 6.

4. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items that lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement.

5. If the INTO phrase is specified, the record being read is moved from the record area to the area specified by data-name-1, according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with data-name-1 is evaluated after the record has been read and immediately before it is moved to the data item. Data-name-1 must not be defined in the FD entry for the file.

6. When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with data-name-1.

7. If, at the time of execution of a READ statement, the position of the current record pointer for that file is undefined, the execution of that READ statement is unsuccessful.

8. If, at the time of the execution of a READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the READ is unsuccessful.

9. When the AT END condition is recognized, the following actions are taken in the specified order:

   - A value is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition.

   - If the AT END phrase is specified in the statement causing the condition, control is transferred to the imperative-statement. Any USE procedure specified for this file is not executed.

   - If the AT END phrase is not specified, then a USE procedure must be specified for this file, and that procedure is executed.

   When the AT END condition occurs, execution of the input-output statement that caused the condition is unsuccessful.

10. Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined.

11. When the AT END condition has been recognized, a READ statement for that file must not be executed without first executing a successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

12. Prime extension: for sequential disk files containing packed or binary data, the user should specify UNCOMPRESSED in the FD entry for that file.

READY TRACE — PRIME EXTENSION

Function

The READY TRACE statement enables the display of trace information to the user terminal.

Format

READY TRACE.

General Rules

1. After a READY TRACE statement is executed, each time a paragraph or section in the PROCEDURE division is encountered, that paragraph or section name is output to the terminal to provide debugging information. The format printed is:

   trace: { section-name / paragraph-name }

2. READY TRACE must not be used before the first paragraph-name in the PROCEDURE division.

3. The display of trace information can be terminated with execution of the RESET TRACE statement.

Example

When the sample program at the end of this chapter is run with READY TRACE inserted at the start of the PROCEDURE division as shown, the actual flow of program execution is displayed. Output might be:

```
OK, SEG OLDCASH
trace:  010-GET-JOBINFO
ENTER MONTH(ALPHA)
JUNE, 1982
ENTER JOB CODE
25
trace:  020-NEW-DETAIL-PAGE
trace:  150-NEW-PAGE
trace:  150-NEW-PAGE-EXIT
trace:  030-PROCESS-DETAIL
trace:  035-READ-AND-PRINT
trace:  040-EDIT
trace:  050-DEPT-TOTALS
trace:  035-READ-AND-PRINT
```

```
trace:  040-EDIT
trace:  050-DEPT-TOTALS
trace:  040-READ-AND-PRINT
trace:  040-EDIT
trace:  050-DEPT-TOTALS
trace:  035-READ-AND-PRINT
trace:  040-EDIT
trace:  050-DEPT-TOTALS
trace:  035-READ-AND-PRINT
trace:  040-EDIT
trace:  050-DEPT-TOTALS
trace:  035-READ-AND-PRINT
trace:  040-EDIT
trace:  060-REJECTS
trace:  035-READ-AND-PRINT
trace:  040-EDIT
trace:  060-REJECTS
trace:  035-READ-AND-PRINT
trace:  040-EDIT
trace:  060-REJECTS
trace:  035-READ-AND-PRINT
trace:  040-EDIT
trace:  060-REJECTS
trace:  070-TOTALS
trace:  150-NEW-PAGE
trace:  150-NEW-PAGE-EXIT
trace:  080-BALANCE-TOTALS
trace:  150-NEW-PAGE
trace:  150-NEW-PAGE-EXIT
trace:  090-PROCESS-TAPE
IS TAPE OUTPUT DESIRED--ENTER YES OR NO
NO
NO TAPE
     END OF RUN
OK,
```

RELEASE

Function

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

Format

RELEASE record-name [FROM data-name]

Syntax Rule

A RELEASE statement may be used only within an input procedure associated with a SORT statement. The SORT statement must name a file whose SD entry contains the record-name.

For a complete discussion, see Chapter 11, THE SORT-MERGE MODULE.

RESET TRACE — PRIME EXTENSION

Function

This statement turns off the display of trace information.

Format

RESET TRACE.

General Rules

1. The RESET TRACE statement has meaning only after the execution of a READY TRACE statement.

2. RESET TRACE must not be used before the first paragraph-name in the PROCEDURE division.

RETURN

Function

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

Format

RETURN file-name RECORD [INTO data-name]

; AT END imperative-statement

Syntax Rule

A RETURN statement may be used only within an output procedure associated with a SORT statement for a file-name described by an SD entry.

For complete information, see Chapter 11, THE SORT-MERGE MODULE.

REWRITE

Function

The REWRITE statement logically replaces a record existing in a disk file.

Format 1

REWRITE record-name [FROM data-name]

Format 2

REWRITE record-name [FROM data-name]

[; INVALID KEY imperative-statement]

Syntax Rules

1. The record-name and the data-name may refer to the same storage area.

2. The record-name is the name of a logical record in the FILE section and may be qualified.

Note

This discussion of the REWRITE statement includes sequentially-organized files only. See Chapters 12 and 13 for additional information on indexed files and relative files, respectively.

General Rules

1. The file containing record-name must be a disk file and must be open for I-O (in all access methods) prior to execution of a REWRITE statement.

2. The last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. REWRITE logically replaces the record that was accessed by the READ statement.

3. The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.

4. Prime extension: the logical record released by a successful execution of the REWRITE statement is still available in the record area.

5. If the associated file is named in a SAME RECORD AREA clause, the logical record is also available to the program as a record of other files appearing in the SAME RECORD AREA clause, as well as to the file associated with record-name.

6. If the FROM option is used, the information in data-name is moved to the record area prior to the REWRITE.

7. The current record pointer (the conceptual entity that determines the next record to be accessed) is not affected by the execution of a REWRITE statement.

8. The execution of the REWRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated.

9. The INVALID KEY option is not used for sequential files.

10. A sequential file used with REWRITE must be either a COBOL-created file other than a printer file, or any uncompressed file.

SEARCH

Function

The SEARCH statement is used to search a table for a table element that satisfies the specified condition, and to adjust the associated index-name to indicate that table element.

Format 1

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

  [ ; AT END imperative-statement-1]

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

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

Format 2

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

  ; WHEN { data-name-2 { IS EQUAL TO } { data-name-3 }
        {               { IS =       } { literal-1   }
        {                             { arith-expr-1 }
        { condition-name-1                           }

  [ AND { data-name-4 { IS EQUAL TO } { data-name-5 } } ] ...
        {             { IS =       } { literal-2   } }
        {                           { arith-expr-2 } }
        { condition-name-2                           }

  { imperative-statement-2 }
  { NEXT SENTENCE          }
```

Syntax Rules

1. In both Formats 1 and 2, data-name-1 must not be subscripted or indexed, but its description must contain an OCCURS clause and an INDEXED BY clause.

2. If data-name-2 is specified, it must be described as USAGE IS INDEX, or as a numeric elementary item without any positions to the right of the assumed decimal point.

3. In Format 1, conditions 1 and 2 can be any condition, including the CORRESPONDING relation condition.

4. In Format 2, the description of data-name-1 must contain the KEY IS phrase in its OCCURS clause.

Note

A complete discussion of the SEARCH verb is presented in Chapter 10, TABLE HANDLING.

SEEK — PRIME EXTENSION

Function

SEEK specifies a disk record to be accessed.

Format

SEEK file-name RECORD

General Rules

1. SEEK is treated as documentation only in Prime COBOL 74. It is included only for compatibility with programs transported from non-Prime systems.

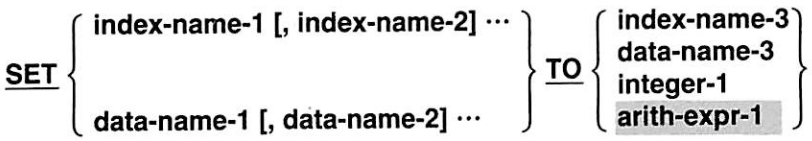
2. The file-name must be defined by a file-description-entry in the DATA division.

SET

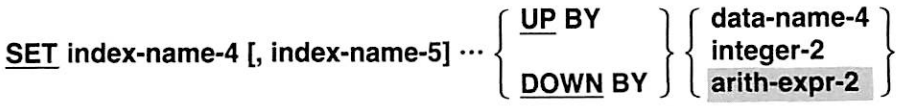
Function

The SET statement establishes reference points for table-handling operations by setting index-names associated with table elements.

Format 1



Format 2



The SET statement is discussed in Chapter 10, TABLE HANDLING.

SKIP -- PRIME EXTENSION

Function

The SKIP statement directs the compiler to place blank lines in the program listing.

Format

SKIPn

Syntax Rule

The letter n represents the number 1, 2, or 3 written as one word with SKIP.

General Rule

The number after SKIP causes one, two, or three blank lines to be inserted in the program listing, after the line containing SKIP.

Example

The following program contains SKIP1 in line 4 and SKIP3 in line 6 of the source.

```
Source File:  <OPERSY>ANNE.K>NEWCBL>SIZECK.CBL
Compiled on: FRI, SEP 03 1982 at 08:47 by: CBL rev x 06/09/82.09:07
:44.Wed
Options are:  LISTING OPTIMIZE U(PPER)CASE

    1           IDENTIFICATION DIVISION.
    2           PROGRAM-ID. SIZECK.
    3           AUTHOR.  ANNE LADD.

    5           DATE-COMPILED. 820903.08:47:11.



    7           ENVIRONMENT DIVISION.
    8           CONFIGURATION SECTION.
    9           SOURCE-COMPUTER. PRIME.
   10           OBJECT-COMPUTER.
   11           INPUT-OUTPUT SECTION.
   12           FILE-CONTROL.
   13               SELECT OPTIONAL DISK-FILE ASSIGN TO PFMS.
```

SORT

Function

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

Format

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

[ ON { ASCENDING  } KEY data-name-3 [, data-name-4] ... ] ...
     { DESCENDING }

[COLLATING SEQUENCE IS alphabet-name]

{ INPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ] }
                                     { THRU    }
{ USING file-name-2 [, file-name-3] ...                            }

{ OUTPUT PROCEDURE IS section-name-3 [ { THROUGH } section-name-4 ] }
                                       { THRU    }
{ GIVING file-name-4                                               }
```

Note

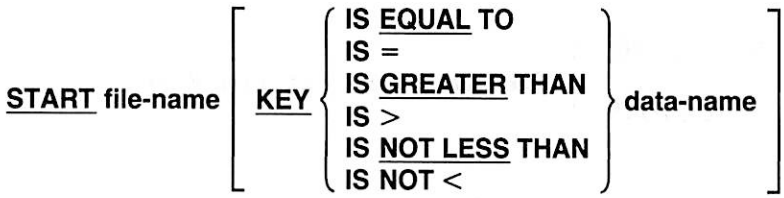
A complete discussion of the SORT statement is presented in Chapter 11, THE SORT-MERGE MODULE.

START

Function

The START statement provides a basis for logical positioning within an indexed or relative file for subsequent sequential or dynamic retrieval of records.

Format



[; INVALID KEY imperative-statement]

START is discussed in Chapters 12 and 13 on indexed and relative I-O.

STOP

Function

The STOP statement is used to terminate or delay execution of the object program.

Format

STOP { **RUN** / **literal** }

Syntax Rule

1. The literal must be a numeric unsigned integer, nonnumeric literal, or any figurative constant without the keyword ALL.

2. If a STOP RUN statement appears in a consecutive sequence of imperative statements within a sentence, it should appear as the last statement in that sequence.

General Rules

1. STOP RUN terminates execution of a program, returning control to the operating system.

2. If STOP RUN is used in a called program, execution halts. Control is not returned to the calling program.

3. If STOP literal is specified, the literal is displayed on the terminal, and execution is suspended. Execution is resumed at the next executable statement in sequence after the carriage return is pressed. Presumably, the operator performs a function suggested by the contents of the literal prior to resuming program execution.

4. A CLOSE statement should be executed before a STOP RUN statement if any files were open in the program.

STRING

Function

The STRING statement concatenates the partial or complete contents of two or more data items.

Format

```
STRING { data-name-1 } [ , data-name-2 ] ... DELIMITED BY { data-name-3 }
       { literal-1   }   [ , literal-2 ]                   { literal-3   }
                                                           { SIZE        }

[ , { data-name-4 } [ , data-name-5 ] ... DELIMITED BY { data-name-6 } ] ...
    { literal-4   }   [ , literal-5 ]                  { literal-6   }
                                                       { SIZE        }

INTO data-name-7 [WITH POINTER data-name-8]

[ ; ON OVERFLOW imperative-statement]
```

Syntax Rules

1. Each literal may be any figurative constant (without the optional word ALL).

2. All literals must be described as nonnumeric literals. All data-names, except data-name-8, must be described implicitly or explicitly as USAGE IS DISPLAY.

3. The operand after INTO, data-name-7, must represent an elementary alphanumeric data item without editing symbols or the JUSTIFIED clause.

4. The POINTER operand, data-name-8, must represent an elementary numeric integer data item of sufficient size to contain a value equal to the size of data-name-7 + 1. The symbol P may not be used in the PICTURE character-string of data-name-8.

5. Where data-name-1, data-name-2, or data-name-3 is an elementary numeric data item, it must be described as an integer without the symbol P in its PICTURE character-string.

6. Operands of STRING must be less than 32K bytes in length.

General Rules

1. All references to data-names 1-3 and literals 1-3 apply equally to data-names 4-6 and literals 4-6, respectively, and to all recursions thereof.

2. The items referred to by data-name-1, literal-1, data-name-2, and literal-2 are the sending items. Data-name-7 represents the receiving item.

3. The operands of DELIMITING (literal-3, data-name-3) indicate the character(s) delimiting the move. If the SIZE phrase is used, the complete sending item is moved. When a figurative constant is used as the delimiter, it stands for a single-character nonnumeric literal.

4. When a figurative constant is specified as literal-1, literal-2, literal-3, it refers to an implicit one-character data item whose usage is DISPLAY.

5. When the STRING statement is executed, the transfer of data is governed by the following rules:

   - Characters from the sending items are transferred to data-name-7 in accordance with the rules for alphanumeric to alphanumeric moves, except that no space-filling is provided.

   - If the DELIMITED phrase is specified without the SIZE phrase, the contents of the sending items are transferred to the receiving data item. This occurs in the sequence specified in the STRING statement, beginning with the leftmost character and continuing from left to right until the end of the data item is reached, or until the character(s) specified by literal-3 or by the contents of data-name-3 are encountered. The character(s) specified by literal-3 or by the data item referenced by data-name-3 are not transferred.

   - If the DELIMITED phrase is specified with the SIZE phrase, the entire contents of literal-1, literal-2, or data-name-1, data-name-2, are transferred. The transfer proceeds in the sequence specified in the STRING statement to data-name-7, until all data has been transferred or the end of the data item referenced by data-name-7 has been reached.

6. If the POINTER phrase is specified, the programmer is responsible for setting the initial value of data-name-8. The initial value must not be less than 1.

7. If the POINTER phrase is not specified, the following general rules apply as if the user had specified data-name-8 with an initial value of 1.

8. When characters are transferred to the receiving item (data-name-7), the transfer behaves as though characters were moved, one at a time, from the sending item to the character position of data-name-7 designated by the value of data-name-8. Data-name-8 is increased by one prior to the move of the next character. The value of data-name-8 is changed during execution of the STRING statement only by the behavior specified above.

9. At the end of execution of the STRING statement, only the portion of data-name-7 that was referenced during the execution of the STRING statement is changed. All other portions of data-name-7 will contain data that was present before this execution of the STRING statement.

10. Data transfer to data-name-7 terminates when the value in data-name-8 is either less than 1, or exceeds the number of character positions in data-name-7. Such termination may occur at any point at or after initialization of the STRING statement. If termination occurs as a result of such a condition, the imperative statement in an ON OVERFLOW phrase is executed, if specified.

11. If the ON OVERFLOW phrase is not specified when the conditions described in General Rule 10 above are encountered, control is transferred to the next executable statement.

Note

Prime restriction: the number of sending items may not exceed five.

Example

The following program concatenates two strings to form a file-name.

```
Source File: <MISCE2>ANNE.F>STRING1
Compiled on: 810303  at: 15:27   by: NEW COBOL Rev 18.0.5  <Feb 12,
 1981>
Options: LISTING OPTIMIZE UCASE

    1          ID DIVISION.
    2          PROGRAM-ID. USTRING.
    3          ENVIRONMENT DIVISION.
    4          CONFIGURATION SECTION.
    5          SOURCE-COMPUTER. PRIME-750.
    6          OBJECT-COMPUTER. PRIME-750.
```

```
 7          DATA DIVISION.
 8          WORKING-STORAGE SECTION.
 9          77  FILE-NUMBER-WS          PIC X(1) VALUE '0'.
10          77  FILE-PREFIX-WS          PIC X(8) VALUE '       F'.
11          77  NEWFILENAME-WS          PIC X(9) VALUE 'F1     '.
12          77 POINTER-WS               PIC 999 VALUE 1.
13          PROCEDURE DIVISION.
14          010-STRING.
15              DISPLAY 'USER NAME: '  ACCEPT FILE-PREFIX-WS.
16              DISPLAY 'RUN NO: '  ACCEPT FILE-NUMBER-WS.
17              STRING FILE-PREFIX-WS, FILE-NUMBER-WS DELIMITED
18                  BY ' ' INTO NEWFILENAME-WS
19                  ON OVERFLOW DISPLAY 'FILENAME MAY ONLY BE 8
20          -       'CHARACTERS LONG.   PLEASE START AGAIN'.
21              DISPLAY 'OUTPUT FILE WILL BE ' NEWFILENAME-WS.
22              STOP RUN.
```

When this program is executed (supposing a runfile named STRING.SEG), the following screen dialog might take place.

```
OK, SEG STRING
USER NAME:
ANNE
RUN NO:
1
OUTPUT FILE WILL BE ANNE1
OK,
```

SUBTRACT

Function

The SUBTRACT statement subtracts one or more numeric data items from a specified item and stores the difference.

Format 1

```
SUBTRACT { data-name-1 } [ , data-name-2 ] ...
         { literal-1   }   { , literal-2 }
         { arith-expr-1}   { , arith-expr-2 }

  FROM data-name-3 [ROUNDED] [, data-name-4 [ROUNDED]] ...

  [; ON SIZE ERROR imperative-statement]
```

Format 2

```
SUBTRACT { data-name-1 } [ , data-name-2 ] ...
         { literal-1   }   { , literal-2 }
         { arith-expr-1}   { , arith-expr-2 }

  FROM { data-name-3 }
       { literal-3   } GIVING data-name-6 [ROUNDED]
       { arith-expr-3}

  [, data-name-7 [ROUNDED]] ...

  [; ON SIZE ERROR imperative-statement]
```

Format 3

```
SUBTRACT { CORRESPONDING } data-name-1
         { CORR          }

  FROM data-name-2 [ROUNDED]

  [; ON SIZE ERROR imperative-statement]
```

Syntax Rules

1. Each data-name must refer to a numeric elementary item, except that data-name-5, data-name-6, and so on (following GIVING) may be elementary numeric edited items.

2. Each literal must be a numeric literal.

3. The maximum size of each operand is 18 decimal digits. If all receiving data items were to be superimposed aligned by their decimal points, their composite could not exceed 18 decimal digits in length. This rule is ignored if GIVING is used.

4. In Format 3, both data-name-1 and data-name-2 must be group items.

5. The use of arithmetic expressions in SUBTRACT statements is a Prime extension.

General Rules

1. The SUBTRACT statement is governed by the rules for the GIVING, CORRESPONDING, ROUNDED, and SIZE ERROR clauses at the start of this chapter, by the rules for arithmetic statements and algebraic signs in Chapter 4, and by the rules for Arithmetic Statements in the PROCEDURE Division at the beginning of this chapter.

2. In Format 1, the effect of the SUBTRACT statement is to sum the values of all the operands that precede FROM, and then to subtract that sum from the value of each of the operands following FROM. The result is stored in each of the operands following FROM.

3. In Format 2, all operands preceding FROM are added together, the resulting sum is subtracted from data-name-3, arithmetic-expression-3, or literal-3, and the result is stored in each of the operands following GIVING.

4. In Format 3, elementary items subordinate to data-name-1 are subtracted from and stored into the matching elementary items subordinate to data-name-2.

5. The SIZE ERROR statement is executed if the result is too large for its field.

UNSTRING

Function

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

Format

UNSTRING data-name-1

```
[ DELIMITED BY [ALL] { data-name-2 } [ , OR [ALL] { data-name-3 } ] ... ]
                     { literal-1   }               { literal-2   }
```

INTO data-name-4 [, DELIMITER IN data-name-5] [, COUNT IN data-name-6]

[, data-name-7 [, DELIMITER IN data-name-8] [, COUNT IN data-name-9]] ...

[WITH POINTER data-name-10] [TALLYING IN data-name-11]

[; ON OVERFLOW imperative-statement]

Syntax Rules

1. Each literal must be a nonnumeric literal. In addition, each literal may be any figurative constant without the optional word ALL. (The ALL phrase option of UNSTRING is not the figurative constant ALL.)

2. The items represented by data-name-1, data-name-2, data-name-3, data-name-5, and data-name-8 must be described, implicitly or explicitly, as alphanumeric.

3. The items represented by data-name-4 and data-name-7 may be described as either alphabetic (except that the symbol B may not be used in their picture-strings), alphanumeric, or numeric (except that the symbol P may not be used in their picture-strings). They must be described with USAGE IS DISPLAY.

4. The items represented by data-name-6, data-name-9, data-name-10, and data-name-11 must be described as elementary numeric integer data items (except that the symbol P may not be used in their picture-strings). No binary items (COMP-1, COMP-2, or COMP) are allowed for these fields.

5. No data-name may name a level-88 entry.

6. The DELIMITER IN phrase and the COUNT IN phrase may be specified only if the DELIMITED BY phrase is specified.

7. Operands of UNSTRING may not be longer than 32767 bytes in length.

General Rules

1. All references to data-name-2, literal-1, data-name-4, data-name-5, and data-name-6 apply equally to data-name-3, literal-2, data-name-7, data-name-8, and data-name-9, respectively.

2. Data-name-1 represents the sending area.

3. Data-name-4 represents the receiving area. Data-name-5 represents the receiving area for delimiters.

4. Literal-1 or the data item referenced by data-name-2 specifies a delimiter.

5. Data-name-6 represents the count of the number of characters within data-name-1 isolated by the delimiters for the move to data-name-4. This value does not include a count of the delimiter character(s).

6. The data item referenced by data-name-10 contains a value that indicates a relative character position within the area defined by data-name-1.

7. The data item referenced by data-name-11 is a counter that records the number of data items acted upon during the execution of an UNSTRING statement.

8. When a figurative constant is used as the delimiter, it stands for a single-character nonnumeric literal.

   When the ALL phrase is specified, two or more contiguous occurrences of literal-1 (figurative constant or not), or of the contents of data-name-2, are treated as only one occurrence. This occurrence is moved to the receiving data item (data-name-4) according to the rules for the DELIMITER IN phrase in General Rule 13 below.

9. When an examination encounters two contiguous delimiters, the current receiving area is either space or zero filled according to the description of the receiving area.

10. The literal-1, or the contents of the data item referenced by data-name-2, can contain any character in the computer's character set.

11. Each literal-1 or data-name-2 represents one delimiter. When a delimiter contains two or more characters, all of the characters must be present in contiguous positions of the sending item and in the order given to be recognized as a delimiter.

12. When two or more delimiters are specified in the DELIMITED BY phrase, an OR condition exists between them. Each delimiter is compared to the sending field. If a match occurs, the character(s) in the sending field is considered to be a single delimiter. No character(s) in the sending field can be considered as part of more than one delimiter.

    Each delimiter is applied to the sending field in the sequence specified in the UNSTRING statement.

13. When the UNSTRING statement is initiated, the current receiving area is the data item referenced by data-name-4. Data is transferred from data-name-1 to data-name-4 according to the following rules:

    - If the POINTER phrase is specified, the string of characters referenced by data-name-1 is examined beginning with the relative character position indicated by the contents of data-name-10. If the POINTER phrase is not specified, the string of characters is examined beginning with the leftmost character position.

    - If the DELIMITED BY phrase is specified, the examination proceeds, left to right, until a delimiter specified by either literal-1 or data-name-2 is encountered. (See General Rule 11.) If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the receiving area. However, if the sign of the receiving area is defined as occupying a separate character position, the number of characters examined is one less than the size of the current receiving area.

      If the end of data-name-1 is encountered before the delimiting condition is met, the examination terminates with the last character examined.

    - The characters thus examined (excluding any delimiting characters), are treated as an elementary alphanumeric data item, and are moved into the current receiving area according to the rules for the MOVE statement.

- If the DELIMITER IN phrase is specified, the delimiting character(s) are treated as an elementary alphanumeric data item and are moved into data-name-5 according to the rules for the MOVE statement. If the delimiting condition is the end of data-name-1, then data-name-5 is space-filled.

- If the COUNT IN phrase is specified, a value equal to the number of characters thus examined (excluding the delimiter character(s), if any) is moved into data-name-6 according to the rules for an elementary move.

- If the DELIMITED BY phrase is specified, the string of characters is further examined, beginning with the first character to the right of the delimiter. If the DELIMITED BY phrase is not specified, the string of characters is further examined, beginning with the character to the right of the last character transferred.

- After data is transferred to data-name-4, the current receiving area is data-name-7. The behavior described in the preceding four paragraphs is repeated until either all the characters are exhausted in data-name-1, or until there are no more receiving areas.

14. The initialization of the contents of the data items associated with the POINTER phrase or the TALLYING phrase is the responsibility of the user.

15. The contents of data-name-10 will be incremented by one for each character examined in data-name-1. When the execution of an UNSTRING statement with a pointer phrase is completed, data-name-10 will contain a value equal to the initial value, plus the number of characters examined in data-name-1.

16. When the execution of an UNSTRING statement with a TALLYING phrase is completed, the contents of data-name-11 will be a value equal to its initial value, plus the number of data receiving items acted upon.

17. Either of the following situations causes an overflow condition:

    - An UNSTRING is initiated, and the value of data-name-10 is less than 1 or greater than the size of data-name-1.

    - During execution of an UNSTRING statement, all data receiving areas have been acted upon, and data-name-1 contains characters that have not been examined.

18. When an overflow condition exists, the UNSTRING operation is terminated. If an ON OVERFLOW phrase has been specified, the imperative-statement is executed. If the ON OVERFLOW phrase is not specified, control is transferred to the next executable statement.

19. The evaluation of subscripting and indexing for the data-names is as follows:

    - Any subscripting or indexing associated with data-name-1, data-name-10, or data-name-11 is evaluated only once, immediately before any data is transferred as the result of the execution of the UNSTRING statement.

    - Any subscripting or indexing associated with data-name-2 through data-name-6 is evaluated immediately before the transfer of data into the respective data item.

Note

Prime restriction: no more than five data-names may be used as receiving items.

Example

```
 ID DIVISION.
 PROGRAM-ID.  INPUT1.
 AUTHOR. GJK.
 REMARKS.  THIS IS A SUBPROGRAM THAT ACCEPTS THREE FIELDS.  THE FIRST
     FIELD (CALL-INPUT) CONTAINS THE NUMERIC DATA (LEFT-JUSTIFIED) THAT
     WAS ACCEPTED FROM THE KEYBOARD.  THE DATA WILL BE RETURNED TO THE
     CALLING PROGRAM IN THE SECOND FIELD (CALL-RECEIVE), RIGHT-
     JUSTIFIED IF THE FIRST FIELD IS IN CORRECT FORMAT.  OTHERWISE
     AN ERROR CODE IS RETURNED IN THE THIRD FIELD (CALL-ERROR-CODE).
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   AMOUNT-BEFORE-UNSTRING                PIC X(20).
 01   UNSTRING-FIELDS.
      05  UN-AMOUNT-1                       PIC 9(16).
      05  UN-AMOUNT-2                       PIC X(2).
 01   AMOUNT-ALIGNED   REDEFINES UNSTRING-FIELDS  PIC 9(16)V99.
 01   AMOUNT-TEST                           PIC X(20).
 01   INS-TALLY                             PIC 99.
*
 LINKAGE SECTION.
 01   CALL-INPUT                            PIC X(20).
 01   CALL-RECEIVE                          PIC 9(16)V99.
 01   CALL-ERROR-CODE                       PIC 9.
*
```

```
PROCEDURE DIVISION USING CALL-INPUT, CALL-RECEIVE, CALL-ERROR-CODE.
050-MAIN.
    MOVE ZEROS TO UN-AMOUNT-1, INS-TALLY.
    MOVE SPACES TO UN-AMOUNT-2.
    PERFORM 100-EDIT-AMOUNT.
    IF CALL-ERROR-CODE NOT EQUAL 0 NEXT SENTENCE
       ELSE PERFORM 200-PREPARE-FOR-UNSTRING,
            PERFORM 250-ALIGN-AMOUNT-WITH-UNSTRING,
            MOVE AMOUNT-ALIGNED TO CALL-RECEIVE.
    EXIT PROGRAM.
100-EDIT-AMOUNT.
    IF CALL-INPUT EQUAL SPACES
       MOVE 1 TO CALL-ERROR-CODE
    ELSE PERFORM 150-IS-AMOUNT-NUMERIC.
150-IS-AMOUNT-NUMERIC.
    MOVE CALL-INPUT TO AMOUNT-TEST.
    INSPECT AMOUNT-TEST TALLYING INS-TALLY FOR ALL '.'.
    IF INS-TALLY EQUAL 0  MOVE 2 TO CALL-ERROR-CODE
        ELSE INSPECT AMOUNT-TEST REPLACING ALL SPACES BY ZEROES,
             INSPECT AMOUNT-TEST REPLACING FIRST '.' BY ZERO
             IF AMOUNT-TEST IS NUMERIC NEXT SENTENCE,
                ELSE MOVE 1 TO CALL-ERROR-CODE.
200-PREPARE-FOR-UNSTRING.
    INSPECT CALL-INPUT REPLACING ALL SPACES BY ZEROES.
    MOVE CALL-INPUT TO AMOUNT-BEFORE-UNSTRING.
250-ALIGN-AMOUNT-WITH-UNSTRING.
    UNSTRING AMOUNT-BEFORE-UNSTRING DELIMITED BY '.'
        INTO UN-AMOUNT-1, UN-AMOUNT-2.
```

USE

Function

The USE statement specifies procedures for input-output error handling.

Format 1

```
USE AFTER STANDARD { EXCEPTION } PROCEDURE ON { file-name-1 [, file-name-2] ... }
                   { ERROR     }              { INPUT                           }
                                              { OUTPUT                          }
                                              { I-O                             }
                                              { EXTEND                          }.
```

Format 2

```
USE AFTER STANDARD { EXCEPTION } PROCEDURE ON { file-name-1 [, file-name-2] ... }
                   { ERROR     }              { INPUT                           }
                                              { OUTPUT                          }
                                              { I-O                             }.
```

Syntax Rules

1. A USE statement, when present, must immediately follow a section header in the DECLARATIVES section, separated from it by a period and a space. The remainder of the section must consist of zero, one, or more paragraphs that define the procedures to be used.

2. The USE statement itself is never executed; rather, it defines the conditions for the execution of the following paragraphs.

3. The same file-name may appear in more than one USE statement, in a different specific arrangement of the format. Appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE procedure.

4. The words EXCEPTION and ERROR are interchangeable.

5. The files implicitly or explicitly referenced in a USE statement need not all have the same organization or access.

6. Format 1 is for use with sequential files only.

General Rules

1. The paragraphs introduced by USE are executed (implicitly, by the perform mechanism) after the standard I-O recovery procedures for the files designated, or after the invalid key condition arises on a statement lacking the INVALID KEY clause, or on recognition of end of file when AT END has not been specified in the input-output statement. They are also invoked if the program attempts to read a record that has been locked by another user.

2. After execution of a USE procedure, control is returned to the statement following the I-O statement whose execution resulted in invoking the USE procedure.

3. With a USE procedure, there must not be any reference to any nondeclarative procedure. Conversely, in the nondeclarative portion there must not be any reference to procedure-names that appear in the declarative portion, except that PERFORM statements may refer to a USE statement or to the procedures associated with it.

Example

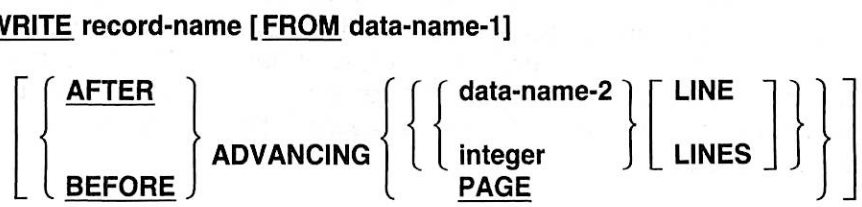
An example of USE in DECLARATIVES sections for disk and tape files is given in the sample program at the end of this chapter.

WRITE

Function

The WRITE statement releases a logical record for an output or I-O file. It can also be used for vertical positioning of lines within a logical page.

Format 1



Format 2

WRITE record-name [FROM data-name-1]

[; INVALID KEY imperative-statement]

Syntax Rules

1. Format 1 can be used only for sequential files.
2. Format 2 can be used only for relative and indexed files.
3. The record-name and data-name-1 may refer to the same storage area.
4. The record-name is the 01-level record-name of a logical record, described in a record-description-entry in the FILE section of the DATA division. It may be qualified.
5. When data-name-2 is used in the ADVANCING phrase, it must be the name of an elementary integer data item.
6. The integer or the value of the data item referenced by data-name-2 must be between 0 and 62 inclusive.

General Rules

Note

For a detailed discussion of indexed and relative writes, see Chapters 12 and 13.

1. For both WRITE statement formats, the associated file must be open as OUTPUT, I-O, or EXTEND.

2. Prime extension: the logical record released by the execution of the WRITE statement is still available in the record area.

3. If the associated file is named in a SAME RECORD AREA clause, the logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as well as to the file associated with record-name.

4. In Format 1, if the FROM option is used, the information is moved to the record area prior to the WRITE. If the data being moved is longer than the receiving field, the data is truncated to the size of the receiving field. If the receiving field is longer than the data, the remaining area is filled with spaces.

   After execution of the WRITE statement, the information in the area referenced by data-name-1 is still available.

5. The current record pointer (the conceptual entity that determines the next record to be accessed) is unaffected by the execution of a WRITE statement.

6. The execution of the WRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated.

7. The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

8. The number of character positions on a disk required to store a logical record in a file may or may not be equal to the number of character positions defined by the logical description of that record in the program.

9. The execution of the WRITE statement releases a logical record to the file system.

10. In Format 1, if the ADVANCING option is used, print control spacing is indicated. The first position in the record must be reserved as FILLER for the print control character being generated.

    - If the BEFORE option is used, a line is written before advancing.
    - If the AFTER option is used first, spacing occurs, and then the line is written.
    - Data-name-2 LINE(s) is the number of spacing lines required between data lines. The value of data-name-2 may be 0 to 62.

    If the ADVANCING option is not used, the default is one line.

11. In Format 1, the significance of the integer is as described in Table 8-3.

Table 8-3
Carriage Control Integer Values

Integer	Carriage Control Actions
0	Overprinting
1	Single spacing
2	Double spacing
3	Triple spacing
4	4-line spacing
5	5-line spacing
6	6-line spacing
.	
.	
.	
62	62-line spacing
PAGE	Skips to top of new page

12. If PAGE is specified, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the next logical page.

    If PAGE has no meaning in conjunction with a specific device, then advancing is provided as if the user had specified BEFORE or AFTER (depending on the phrase used) ADVANCING 1 LINE.

13. When an attempt is made to write beyond the externally defined boundaries of a sequential file, an exception condition exists and the contents of the record area are unaffected. The following action takes place:

    - The value of the FILE STATUS data item, if any, of the associated file is set to a value indicating a boundary violation.
    - If a USE declarative is explicitly or implicitly specified for the file, that declarative procedure will then be executed.
    - If a USE declarative is not explicitly or implicitly specified for the file, the result is undefined.

EXAMPLE

This example forms one program with the examples at the end of Chapters 5, 6, and 7.

```
       PROCEDURE DIVISION.
      *
       DECLARATIVES.
       INPUT-ERROR SECTION. USE AFTER ERROR PROCEDURE ON DISK-FILE.
       FIRST-PARAGRAPH.
           DISPLAY  '**** I-O ERROR ON ENTRY: ***'.
           DISPLAY  ENTRY-DETAIL, CLOSE DISK-FILE, PRINT-FILE.
           STOP RUN.
       TAPE-ERROR SECTION. USE AFTER ERROR PROCEDURE ON TAPE-FILE.
       SECOND-PARAGRAPH.
           DISPLAY '**** I-O ERROR ON TAPE OUTPUT ***'.
           CLOSE DISK-FILE, PRINT-FILE.  STOP RUN.
       END DECLARATIVES.
      *
       001-BEGIN.
           READY TRACE.
           OPEN INPUT DISK-FILE, OUTPUT PRINT-FILE.
           PERFORM 010-GET-JOBINFO.
           PERFORM 020-NEW-DETAIL-PAGE.
           PERFORM 030-PROCESS-DETAIL.
           PERFORM 070-TOTALS.
           PERFORM 080-BALANCE-TOTALS.
           PERFORM 090-PROCESS-TAPE.
           CLOSE DISK-FILE, PRINT-FILE.
           DISPLAY '     END OF RUN'.
           STOP RUN.
      *
       010-GET-JOBINFO.
           DISPLAY 'ENTER MONTH(ALPHA)'. ACCEPT VARIABLE-MONTH.
           DISPLAY 'ENTER JOB CODE'. ACCEPT JOB-CODE.
           IF NOT CORRECT-CODE DISPLAY 'WRONG CODE',
               CLOSE DISK-FILE, PRINT-FILE, STOP RUN.
      *
       020-NEW-DETAIL-PAGE.
           MOVE '      DETAIL LIST       ' TO VARIABLE-HEADING.
           PERFORM 150-NEW-PAGE THRU 150-NEW-PAGE-EXIT.
           MOVE SPACES TO PRINT-LINE.
           MOVE ' DATE              VENDOR                CHECK
      -    '      ACCOUNT       AMOUNT' TO PRINT-LINE.
           WRITE PRINT-LINE AFTER ADVANCING VARIABLE LINES.
           ADD 4 TO LINECOUNT.
           EJECT
       030-PROCESS-DETAIL.
           READ DISK-FILE AT END MOVE 'Y' TO NO-MORE-RECORDS,
               DISPLAY 'INPUT FILE WAS EMPTY',
               CLOSE PRINT-FILE, DISK-FILE, STOP RUN.
           PERFORM 035-READ-AND-PRINT UNTIL NO-MORE-RECORDS = 'Y'.
           EXIT.
```

```
 035-READ-AND-PRINT.
      MOVE 0 TO ERR-CODE.
      PERFORM 040-EDIT.
      IF ERR-CODE NOT = 0 PERFORM 060-REJECTS,
      ELSE PERFORM 050-DEPT-TOTALS.
      MOVE 1 TO VARIABLE.
      MOVE CORRESPONDING ENTRY-DETAIL TO PRINT-DETAIL.
      MOVE ENTRY-ACCT-NO TO PRINT-ACCT-NO.
      MOVE ENTRY-AMOUNT TO PRINT-AMOUNT.
      WRITE PRINT-LINE FROM PRINT-DETAIL AFTER ADVANCING
          1 LINE.
      ADD 1 TO LINECOUNT.
      IF LINECOUNT > 50 PERFORM 020-NEW-DETAIL-PAGE.
*****************************************************************
* READ ALL SUBSEQUENT ENTRIES.                                  *
*****************************************************************
      READ DISK-FILE AT END MOVE 'Y' TO NO-MORE-RECORDS.
      EXIT.
*
 040-EDIT.
*****************************************************************
* ONLY ONE ERROR IS FLAGGED FOR EACH REJECT                     *
*****************************************************************
      MOVE 0 TO ERR-CODE.
      IF ENTRY-ACCT-NO NOT NUMERIC  MOVE 1 TO ERR-CODE.
      IF ENTRY-ACCT-NO LESS THAN 100 OR GREATER THAN 449,
          MOVE 2 TO ERR-CODE.
      IF ENTRY-AMOUNT NOT NUMERIC  MOVE 3 TO ERR-CODE.
      IF ENTRY-MONTH OF ENTRY-DETAIL NOT NUMERIC MOVE 4 TO
          ERR-CODE.
      IF ENTRY-CHECK-NO OF ENTRY-DETAIL NOT NUMERIC,
            MOVE 7 TO ERR-CODE.
*
 050-DEPT-TOTALS.
*****************************************************************
* MAKE CROSS-TOTAL AS CHECK,                                    *
* FIND HOME-ACCOUNT FOR EACH ACCOUNT NUMBER.                    *
* ADD ENTRY-AMOUNT TO HOME DEPARTMENT TOTAL.                    *
*****************************************************************
      ADD ENTRY-AMOUNT TO CROSS-TOTAL.
      IF ENTRY-ACCT-NO LESS THAN 200, ADD ENTRY-AMOUNT TO TOTAL1,
      ELSE  IF ENTRY-ACCT-NO LESS THAN 300 AND ENTRY-ACCT-NO >
                        199, ADD ENTRY-AMOUNT TO TOTAL2,
            ELSE IF ENTRY-ACCT-NO LESS THAN 420 AND ENTRY-ACCT-NO
                      > 300, ADD ENTRY-AMOUNT TO TOTAL3,
                 ELSE IF ENTRY-ACCT-NO LESS THAN 430 AND
                        ENTRY-ACCT-NO > 419, ADD ENTRY-AMOUNT TO
                                             TOTAL4,
                      ELSE IF ENTRY-ACCT-NO  LESS THAN 440 AND
                              ENTRY-ACCT-NO > 429, ADD
                              ENTRY-AMOUNT TO TOTAL5,
                           ELSE IF ENTRY-ACCT-NO > 439, ADD
                                ENTRY-AMOUNT TO TOTAL6.
*
```

```
 060-REJECTS.
*****************************************************************
* MAKE CROSS-TOTAL FOR REJECTS.                                 *
*****************************************************************
     IF ENTRY-AMOUNT NUMERIC, ADD ENTRY-AMOUNT TO REJECT-TOTAL.
     MOVE SPACES TO ERROR-LINE.
     MOVE  ' ** ERROR FOLLOWS **' TO MESSAGE.
     WRITE ERROR-LINE.
     EJECT
 070-TOTALS.
     MOVE 'TOTALS BY ACCOUNT NUMBER' TO VARIABLE-HEADING.
     PERFORM 150-NEW-PAGE THRU 150-NEW-PAGE-EXIT.
     MOVE '           ACCOUNT              TOTAL DISBURSEMENT
-    ' ' TO PRINT-LINE.
     WRITE PRINT-LINE AFTER ADVANCING VARIABLE LINES.
*****************************************************************
* PRINT TOTALS FOR EACH HOME ACCOUNT                            *
*****************************************************************
     MOVE '100' TO HOME-NUMBER.
     MOVE TOTAL1 TO HOME-TOTAL.
     WRITE PRINT-LINE FROM HOME-ACCT-LINE AFTER ADVANCING 1.
     MOVE '200' TO HOME-NUMBER.
     MOVE TOTAL2 TO HOME-TOTAL.
     WRITE PRINT-LINE FROM HOME-ACCT-LINE AFTER ADVANCING 1.
     MOVE '410' TO HOME-NUMBER.
     MOVE TOTAL3 TO HOME-TOTAL.
     WRITE PRINT-LINE FROM HOME-ACCT-LINE AFTER ADVANCING 1.
     MOVE '420' TO HOME-NUMBER.
     MOVE TOTAL4 TO HOME-TOTAL.
     WRITE PRINT-LINE FROM HOME-ACCT-LINE AFTER ADVANCING 1.
     MOVE '430' TO HOME-NUMBER.
     MOVE TOTAL5 TO HOME-TOTAL.
     WRITE PRINT-LINE FROM HOME-ACCT-LINE AFTER ADVANCING 1.
     MOVE '440' TO HOME-NUMBER.
     MOVE TOTAL6 TO HOME-TOTAL.
     WRITE PRINT-LINE FROM HOME-ACCT-LINE AFTER ADVANCING 1.
     MOVE 'REJ' TO HOME-NUMBER.
     MOVE REJECT-TOTAL TO HOME-TOTAL.
     WRITE PRINT-LINE FROM HOME-ACCT-LINE AFTER ADVANCING 1.
     ADD TOTAL1, TOTAL2, TOTAL3, TOTAL4, TOTAL5, TOTAL6,
       REJECT-TOTAL GIVING FINAL-TOTAL.
     MOVE 'FINAL TOTAL          ' TO HOME-NUMBER.
     MOVE FINAL-TOTAL TO HOME-TOTAL.
     WRITE PRINT-LINE FROM HOME-ACCT-LINE AFTER ADVANCING 2.
 080-BALANCE-TOTALS.
     MOVE '      BALANCE RUN       ' TO VARIABLE-HEADING.
     PERFORM 150-NEW-PAGE THRU 150-NEW-PAGE-EXIT.
*****************************************************************
* GOOD ITEMS AND REJECTS ARE ADDED FOR GRAND-TOTAL, WHICH       *
* COMPARED WITH THE FINAL TOTAL (OBTAINED BY ADDING ACCOUNT     *
* TOTALS AND REJECT TOTAL).                                     *
*****************************************************************
     MOVE '             GOOD ITEMS          REJECT TOTAL
-     '        GRAND-TOTAL  ' TO PRINT-LINE.
```

```
           WRITE PRINT-LINE AFTER ADVANCING VARIABLE LINES.
           ADD CROSS-TOTAL, REJECT-TOTAL GIVING GRAND-TOTAL.
           MOVE GRAND-TOTAL TO FIELD-DIFF.
           MOVE REJECT-TOTAL TO FIELD-REJECT.
           MOVE CROSS-TOTAL TO FIELD-TOTAL.
           WRITE PRINT-LINE FROM BALANCE-LINE AFTER ADVANCING 1.
           IF GRAND-TOTAL NOT EQUAL FINAL-TOTAL,
               MOVE '*** TOTALS DO NOT BALANCE ***' TO ERROR-LINE,
               WRITE ERROR-LINE AFTER ADVANCING 2 LINES.
      /    EJECT
       090-PROCESS-TAPE.
           DISPLAY 'IS TAPE OUTPUT DESIRED--ENTER YES OR NO '.
           ACCEPT TAPE-CHOICE.
           IF TAPE-CHOICE = 'yes' OR
              TAPE-CHOICE = 'YES' PERFORM 095-WRITE-TAPE,
           ELSE DISPLAY 'NO TAPE'.
      *
       095-WRITE-TAPE.
           OPEN OUTPUT TAPE-FILE.
           MOVE 1 TO VARIABLE.
           MOVE VARIABLE-MONTH TO TAPE-MONTH.
           WRITE TAPE-LINE FROM TAPE-HEADER AFTER ADVANCING 1.
           ACCEPT JOB-DATE FROM DATE.
           MOVE JOB-DATE TO SAVE-DATE-TAPE.
           MOVE '100' TO SAVE-ACCT-TAPE.
           MOVE TOTAL1 TO SAVE-TOTAL-TAPE.
           WRITE TAPE-LINE FROM SAVE-TAPE  AFTER ADVANCING 1.
           MOVE '200' TO SAVE-TOTAL-TAPE.
           MOVE TOTAL2 TO SAVE-TOTAL-TAPE.
           WRITE TAPE-LINE FROM SAVE-TAPE AFTER ADVANCING 1.
           MOVE '410' TO SAVE-ACCT-TAPE.
           MOVE TOTAL3 TO SAVE-TOTAL-TAPE.
           WRITE TAPE-LINE FROM SAVE-TAPE AFTER ADVANCING 1.
           MOVE '420' TO SAVE-ACCT-TAPE.
           MOVE TOTAL4 TO SAVE-TOTAL-TAPE.
           WRITE TAPE-LINE FROM SAVE-TAPE AFTER ADVANCING 1.
           MOVE '430' TO SAVE-ACCT-TAPE.
           MOVE TOTAL5 TO SAVE-TOTAL-TAPE.
           WRITE TAPE-LINE FROM SAVE-TAPE AFTER ADVANCING 1.
           MOVE '440' TO SAVE-ACCT-TAPE.
           MOVE TOTAL6 TO SAVE-TOTAL-TAPE.
           WRITE TAPE-LINE FROM SAVE-TAPE AFTER ADVANCING 1.
           CLOSE TAPE-FILE.
           PERFORM 095-VERIFY-TAPE.
       095-VERIFY-TAPE.
           DISPLAY 'FIRST TAPE RECORD - VERIFICATION ONLY'.
           OPEN INPUT TAPE-FILE.
           READ TAPE-FILE INTO TAPE-HEADER.
           READ TAPE-FILE.
           DISPLAY TAPE-LINE.
           CLOSE TAPE-FILE.
           EXIT.
      *
```

```
150-NEW-PAGE.
    MOVE PAGECOUNT TO HEADING-PAGE.
    MOVE 2 TO VARIABLE.
    WRITE PRINT-LINE FROM HEADING1 AFTER ADVANCING PAGE.
    WRITE PRINT-LINE FROM HEADING2 AFTER ADVANCING
       VARIABLE LINES.
    WRITE PRINT-LINE FROM HEADING3 AFTER ADVANCING
          VARIABLE LINES.
    ADD 1 TO PAGECOUNT.
    MOVE SPACES TO PRINT-LINE.
    MOVE 8 TO LINECOUNT.
150-NEW-PAGE-EXIT.
```

This program, stored as OLDCASH.CBL, may be compiled, loaded, and executed with the following dialog. (The tape sections are executed in the example at the end of Chapter 14.) The display caused by READY TRACE is given with the discussion of that statement above and is omitted here.

```
OK, CBL OLDCASH -LIST
[CBL  rev XX]
OK, SEG -LOAD
[SEG rev 19.0]
$ LO OLDCASH
$ LI CBLLIB
$ LI
LOAD COMPLETE
$ EXEC

ENTER MONTH(ALPHA)
JUNE, 1982
ENTER JOB CODE
25
IS TAPE OUTPUT DESIRED--ENTER YES OR NO
NO
NO TAPE
     END OF RUN
OK,
```

A sample input file (DISBURSE) is:

```
408080178   ASHTABULA HDWE       4300035476
409080178   CAIRO CHEMICAL       4360002746
410080278   ST.BOTOLPHSTOWN SUPP4200005108
411080278   DOVER MUTUAL         4100034166
412080378   PARIS AUTO           4100015000
413090378   ROME BOATING         4150017982
C82080778   ODESSA SERVICES      4100004670
4500B0778   ANTIOCH SERVALL      4300002580
580080778   BETHLEHEM TAXI       RR00009840
681080778   ATHENS LUMBER        18500036BB
```

A sample output file (DISB01) is:

```
                    MONTHLY CASH DISBURSEMENTS JOURNAL

                          FOR    JUNE, 1982                    PAGE  1

                                DETAIL LIST

DATE          VENDOR                    CHECK      ACCOUNT      AMOUNT
080178        ASHTABULA HDWE            408        430          354.76
080178        CAIRO CHEMICAL            409        436           27.46
080278        ST.BOTOLPHSTOWN SUPP      410        420           51.08
080278        DOVER MUTUAL              411        410          341.66
080378        PARIS AUTO                412        410          150.00
090378        ROME BOATING              413        415          179.82
** ERROR FOLLOWS ** 7
080778        ODESSA SERVICES           C82        410           46.70
** ERROR FOLLOWS ** 4
0B0778        ANTIOCH SERVALL           450        430           25.80
** ERROR FOLLOWS ** 1
080778        BETHLEHEM TAXI            580        RR0           98.40
** ERROR FOLLOWS ** 3
080778        ATHENS LUMBER             681        185           36.BB
```

```
                    MONTHLY CASH DISBURSEMENTS JOURNAL

                          FOR    JUNE, 1982                    PAGE  2

                          TOTALS BY ACCOUNT NUMBER

          ACCOUNT              TOTAL DISBURSEMENT
            100                          00
            200                          00
            410                      671.48
            420                       51.08
            430                      382.22
            440                          00
            REJ                      170.90
          FINAL TOTAL               1275.68
```

```
                    MONTHLY CASH DISBURSEMENTS JOURNAL

                          FOR    JUNE, 1982                    PAGE  3

                                BALANCE RUN

          GOOD ITEMS          REJECT TOTAL          GRAND-TOTAL
           1104.78              170.90                1275.68
```

9

Interprogram Communication

FUNCTION

Interprogram communication allows one program to communicate with another. Control may be transferred from one program to another within a runfile, and both programs may have access to the same data items.

The calling program must have a CALL statement and, if data is to be transferred, a USING clause in the CALL statement. It may also have an ENTER statement for documentation. If data is to be transferred, the called program must have a LINKAGE section for that data, and a USING clause in its PROCEDURE division header. It may also have an EXIT PROGRAM or GOBACK statement. If no GOBACK or EXIT PROGRAM is included, control is transferred back to the calling program after the last statement of the called program is executed.

The example at the end of this chapter illustrates a Prime extension. In it, a program passes a nonlevel-01 structure to a called program. The level number in the LINKAGE section of the called program is 01, though the level number of the passed parameter is not. The size and structure of the two operands are the same.

COBOL programs may call programs written in other Prime languages, if the programs pass arguments of compatible data types. Table 9-1 at the end of this chapter summarizes which data types are compatible. For full information, see the Subroutines Reference Guide for software Rev. 19 and higher.

LINKAGE SECTION

The LINKAGE section describes data, defined elsewhere in a calling program, that is available to a called program.

Format

LINKAGE SECTION.

[**level-77-description-entry**]
[**record-description-entry**] ...

Syntax Rules

1. The LINKAGE section in a program is meaningful if, and only if, the program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.

2. The LINKAGE section describes data made available in memory from another program module, but which is to be referred to in both the calling and the called programs.

3. No space is allocated in a program for data items described in its LINKAGE section. PROCEDURE division references to such items are resolved at load time, by equating the references in the called program to the locations used in the calling program.

4. Data items defined in the LINKAGE section of the called program may be referred to in the PROCEDURE division of that program only if:

   - They are specified as operands of the USING phrase of the PROCEDURE division header or are subordinate to such operands.

   - The called program is under the control of a CALL statement with a USING phrase. (See the example at the close of this chapter.)

5. Any record-description clause in Chapter 7 may be used to describe items in the LINKAGE section with the following exceptions:

   - The VALUE clause may be specified only for level-88 items.

- Each record-name and level-77 name used in the LINKAGE section must be unique (may not be qualified).

- The programmer must ensure proper correspondence between an argument in a CALL statement and the data-name in a USING list of a PROCEDURE header for the called program.

- Items in the LINKAGE section that bear no hierarchic relationship to one another are classified as noncontiguous elementary items, and need not be grouped into records. They may be defined in separate level-77 entries.

  Such entries must include a level-number 77, a data-name, and a PICTURE clause or the USAGE IS INDEX, COMP, COMP-1, or COMP-2 clause.

PROCEDURE DIVISION

Format

PROCEDURE DIVISION [USING data-name-1 [, data-name-2] ···]

Each of the data-name operands is an entry in the LINKAGE section of the called program. Addresses are passed from an external CALL in one-to-one correspondence to the operands in the USING list of the PROCEDURE division header so that data in the calling program may be manipulated in the called program. Corresponding operands must have identical definitions in the DATA division. It is the user's responsibility to assure that the size and structure of each passed operand are the same as those of the LINKAGE SECTION operand to which it corresponds. No check for this equivalence is done at execution time.

The maximum number of operands in a USING statement is listed in Appendix J.

CALL

The CALL statement allows one program to communicate with another. It causes control to be transferred from one object program to another within a runfile.

Format

CALL literal-1 [USING data-name-1 [, data-name-2] ···]

[; ON OVERFLOW imperative-statement]

Syntax Rules

1. The CALL statement appears in the calling program.

2. The literal must be a nonnumeric literal, with a maximum of eight characters.

3. The USING phrase is included in the CALL statement only if there is a USING phrase in the PROCEDURE division header of the called program. Corresponding USING phrases in the calling and the called programs must have the same number of operands. The maximum number of data-names allowed after USING is listed in Appendix J.

4. Each operand in the USING phrase must have been defined as a data item in the FILE section, WORKING-STORAGE section, or LINKAGE section of the calling program. These data-names may be qualified or subscripted.

5. Prime extensions: arguments in a CALL statement may have any level number. They may be subscripted or qualified.

General Rules

1. The execution of a CALL statement transfers control to the called program.

2. A program is in its initial state the first time it is called within a runfile. On all other entries into the called program, the state of the program remains the same as when control last passed from its EXIT statement back to the calling program. This includes all data fields and the status and positioning of all files.

3. Called programs can contain CALL statements. However, a called program must not contain a CALL statement that directly or indirectly calls the calling program.

4. The data-names specified by the USING phrase of the CALL statement represent those data items in a calling program that may be referred to in the called program.

   The order in which the data-names appear in the USING phrases of the two programs is critical; correspondence is positional, not by name. Corresponding operands in the called and calling programs must have the same number of character positions. Corresponding data-names refer to a single set of data that is available to the called and calling programs.

   Any index-names in the calling and called programs always refer to separate indexes.

5. The called program may be written in any language available on a Prime computer.

Note

The literal-1 must be the program-name given in the PROGRAM-ID statement of the called program, not the PRIMOS file-name.

ENTER

Function

The ENTER statement is used for documentation only. It has no effect on the compiler or the compiled program.

Format

ENTER language-name [routine-name]

Syntax Rule

The language-name and routine-name following ENTER may be any programmer-defined words. Each word must contain at least one alphabetic character.

EXIT PROGRAM

Function

The EXIT PROGRAM statement marks the logical end of a called program.

Format

EXIT PROGRAM

Syntax Rules

1. The EXIT PROGRAM statement must appear in a sentence by itself.
2. The EXIT PROGRAM sentence must be the only sentence in the paragraph.

General Rules

1. The execution of an EXIT PROGRAM statement in a called program causes control to be returned to the calling program.
2. An EXIT PROGRAM statement in a program that is not called functions as an EXIT statement.
3. EXIT PROGRAM suppresses any request for interactive file assignment when the -OLD switch is used.

GOBACK — PRIME EXTENSION

Function

The GOBACK statement marks the logical end of a called program.

Format

GOBACK

General Rules

1. In a called program, execution of GOBACK returns control to the calling program.
2. A GOBACK statement in a program that is not called functions as an EXIT statement.
3. GOBACK is a synonym for EXIT PROGRAM.

LOADING AND EXECUTING MORE THAN ONE PROGRAM

To load a runfile containing interdependent programs, follow the steps given in Chapter 3, loading all program object files one after another. The main program must be loaded first. Naming conventions for binary files and runfiles are explained in Chapter 3. An example is given at the end of this chapter.

For object files whose names do not end in .BIN, use only the *Older Loading Procedure* from Chapter 3:

1. SEG
2. LOAD calling-source-file-name.SEG
3. LOAD calling-object-file-name
4. LOAD called-object-file(s)
5. LI CBLLIB
6. If required, LI VSRTLI or other library names
7. LI
8. EXEC or QUIT

The runfile may subsequently be executed with:

SEG calling-source-file-name

For Rev. 18 and higher, if the object file-name ends in .BIN, use the following Default Loading steps from Chapter 3, which create a runfile named program.SEG:

1. SEG -LOAD
2. LOAD calling source-file-name (minus the .CBL suffix)
3. LO called source-file-name(s) (minus .CBL)
4. LI CBLLIB
5. If required, LI VSRTLI or other library names
6. LI
7. EXEC or QUIT

The runfile may subsequently be executed with:

SEG calling-source-file-name

Error Messages

If the SEG utility does not return the message LOAD COMPLETE after the command LI alone is entered (Step 7 or 6 above), then not all required subprograms or libraries have been loaded. Enter MAP 3 to list all unresolved references.

If you attempt to execute a runfile with unresolved references, the system may return a message such as LINKAGE_FAULT$ or POINTER_FAULT$, or may appear to run the program.

EXAMPLE

This example presents two programs. The first, CALLER, exhibits some values, then calls the second, CALLED, which changes those values. CALLER then exhibits the changed values. Finally, the example shows how to load and execute the two programs together. Note that the argument passed between the two programs is defined with level 02 in CALLER, but level 01 in CALLED.

Calling Program

```
Source File:  <OPERSY>ANNE.K>NEWCBL>CALLER.CBL
Compiled on:  SAT, SEP 25 1982 at 13:59  by: CBL rev 9  06/09/82.09:07:
44.Wed
Options are:  LISTING OPTIMIZE U(PPER)CASE

    1            IDENTIFICATION DIVISION.
    2            PROGRAM-ID. CALLER.
    3           *
    4            DATA DIVISION.
    5           *
    6            WORKING-STORAGE SECTION.
    7            01 A1.
    8                02 A2              PIC X VALUE 'A'.
    9                02 A3.
   10                    03 A4          PIC X VALUE 'B'.
   11                    03 A5          PIC X VALUE 'C'.
   12                    03 A6.
   13                       04 A7       PIC X VALUE 'D'.
   14                       04 A8       COMP-2 VALUE -31415.9E-4.
   15                    03 A9          PIC X(5) VALUE 'COBOL'.
   16           *
   17            PROCEDURE DIVISION.
   18                EXHIBIT NAMED A4.
   19                EXHIBIT NAMED A5.
   20                EXHIBIT NAMED A7.
   21                EXHIBIT NAMED A8.
   22                EXHIBIT NAMED A9.
   23                CALL 'CALLED' USING A3.
   24                EXHIBIT NAMED A4.
   25                EXHIBIT NAMED A5.
   26                EXHIBIT NAMED A7.
   27                EXHIBIT NAMED A8.
   28                EXHIBIT NAMED A9.
   29                STOP RUN.

DIAGNOSTIC SUMMARY

ERROR 275  SEVERITY 1  LINE 23  COLUMN 32  [OBSERVATION, SEMANTICS]
The level number of an argument used in a CALL USING statement
should be 01 or 77.
```

Called Program

```
Source File:  <OPERSY>ANNE.K>NEWCBL>CALLED.CBL
Compiled on:  SAT, SEP 25 1982 at 13:59  by: CBL rev 9  06/09/82.09:07:
44.Wed
Options are:  LISTING OPTIMIZE U(PPER)CASE

    1            IDENTIFICATION DIVISION.
    2            PROGRAM-ID. CALLED.
    3           *
    4            DATA DIVISION.
    5           *
    6            LINKAGE SECTION.
    7            01  ARG1.
    8                03 A4                  PIC X.
    9                03 A5                  PIC X.
   10                03 A6.
   11                    04 A7              PIC X.
   12                    04 A8              COMP-2.
   13                04 A9                  PIC X(5).
   14
   15            PROCEDURE DIVISION USING ARG1.
   16                DISPLAY 'ENTERING CALLED'.
   17                MOVE 'X' TO A4.
   18                MOVE 'Y' TO A5.
   19                MOVE 'Z' TO A7.
   20                MOVE 0 TO A8.
   21                MOVE 'NWCBL' TO A9.
   22                GOBACK.
```

Compilation, Loading, and Execution

To run these programs together, follow these steps:

```
OK, CBL CALLER -L

[CBL rev X.X]

ERROR 275  SEVERITY 1  LINE 23  COLUMN 32  [OBSERVATION, SEMANTICS]
The level number of an argument used in a CALL USING statement
should be 01 or 77.

[1 OBSERVATION in program: <OPERSY>ANNE.K>NEWCBL>CALLER.CBL]

OK, CBL CALLED -L

[CBL rev X.X]
OK, SEG -LOAD
[SEG rev X.X]
$ LOAD CALLER
$ LOAD CALLED
$ LI CBLLIB
```

```
$ LI
LOAD COMPLETE
$ Q
OK,
```

When this runfile is executed, the terminal will display the following:

```
OK, SEG CALLER
A4 = B
A5 = C
A7 = D
A9 = -3.1415901184082E+00
A10 = COBOL
ENTERING CALLED
A4 = X
A5 = Y
A7 = Z
A9 =  0.0000000000000E+00
A10 = NWCBL
OK,
```

Table 9-1
Data Type Compatibility in Prime Languages

Generic Unit/PMA	BASIC/VM	COBOL 74	FORTRAN IV	FORTRAN 77	PASCAL	PL1G
1 bit	*	*	*	*	*	(1) BIT BIT(1)
16-bit Halfword	INT	COMP PIC S9 thru S9(4)	(2) INTEGER INTEGER*2 LOGICAL	(2) INTEGER*2 LOGICAL*2	(3) INTEGER Boolean	FIXED BIN FIXED BIN(15)
32-bit Word	INT*4	COMP PIC S9(5) thru S9(9)	INTEGER*4	INTEGER INTEGER*4 LOGICAL LOGICAL*4	(4) Subrange	FIXED BIN(31)
64-bit Double Word	*	COMP PIC S9(10) thru S9(18)	*	*	*	*
32-bit FLOAT single precision	REAL	COMP-1	REAL REAL*4	REAL REAL*4	REAL	FLOAT BINARY FLOAT BIN (23)
64-bit FLOAT double precision	REAL*8	COMP-2	REAL*8	REAL*8	*	FLOAT BIN(47)
Byte string (Max. 32767)	INT	DISPLAY(5) PIC A(n) PIC 9(n) PIC X(n)	INTEGER	(5) CHARACTER *n	(5) ARRAY [1..n] OF CHAR	(5) CHAR(n)
Varying (6) character string	*	(6)	(6)	(6)	(6)	CHAR(n) VARYING (Descriptor)
(7) 48-bits 3 halfwords	*	*	*	*	(8) ^<type>	POINTER
Packed decimal	*	COMP-3	*	*	*	FIXED DECIMAL
Index	*	INDEX	*	*	*	*

* Not available.

Notes to Table 9-1

1. If used for representing true (1) and false (0), negative numbers are true, positive numbers and 0 are false. In PL1G, '1'B is true; if this value is stored in a 16-bit integer, the sign bit is set, giving 100000 octal, or -32768 decimal. False in PL1G may always be represented as decimal 0.

2. LOGICAL data in FORTRAN represents true and false as 1 and 0, respectively.

3. Boolean data in Pascal is represented in 16 bits where the sign bit determines true and false. (A negative sign means true, a positive sign means false.)

4. To define a 32-bit integer in Pascal, see the Pascal Reference Guide for your revision of software.

5. Where n is a constant expression with the program module. This is not a dynamic length.

6. A character-varying string can be simulated in each language indicated, as discussed in the chapter on that language in the Subroutines Reference Guide for Rev. 19 and higher.

7. This implementation of a pointer in PL1G is subject to change; a program that passes pointers or receives them may have to be recompiled, and a program that assumes a particular form or size of pointer data may have to be rewritten.

8. Where <type> is either a user-defined type or a standard Pascal type.

10
Table Handling

DEFINITION

This module defines tables of repeating data items and accesses those items according to their position in the table. Each item may be identified through use of a subscript or an index. Tables of up to eight dimensions may be defined. Ascending or descending keys for a search may be defined.

Tables are defined with the OCCURS clause in the data-description-entry. A major use of tables is that their elements are accessed by position rather than by name. The item that defines a position in a table is called an index if it is defined with INDEXED BY in the OCCURS clause, or a subscript if it is defined elsewhere as an integer data item. In this text, index-name refers to the item defined with INDEXED BY, and index-data-item refers to the item defined with USAGE IS INDEX. Both index items have the index format described in the section on DATA REPRESENTATION AND ALIGNMENT in Chapter 4. The SET verb is the only means of directly assigning a value to an index-name. (The SEARCH and PERFORM verbs with the VARYING option can also be used to change the value of an index-name.) The SEARCH verb is used to search a table item by item.

Some uses for these elements are discussed in the section STRATEGY at the end of this chapter.

The maximum allowable table size is listed in Appendix J.

DATA DIVISION

OCCURS

The OCCURS clause eliminates the need for separate entries for repeated data items. Further, it supplies information required for the application of subscripts or indexes.

Format 1

OCCURS integer-2 TIMES

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

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

Format 2

OCCURS integer-1 TO integer-2 TIMES DEPENDING ON data-name-1

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

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

Syntax Rules

1. The OCCURS clause must not be used in a data-description-entry that:
   - Has a 66, 77, or 88 level-number.
   - Describes an item whose size is variable (i.e., that has a subordinate item containing Format 2 of the OCCURS clause).
2. The maximum OCCURS specification (integer-1 or integer-2) is listed in Appendix J. The minimum OCCURS specification (integer-1) is 1.
3. The data-name-1 cannot be signed.

4. The key (data-name-2) must be either the name of the entry containing the OCCURS clause, or the name of an entry subordinate to this entry.

5. Each data-name-3 must be the name of an entry subordinate to the group item that is the subject of OCCURS.

6. The data-names in the KEY IS phrase must not contain an OCCURS clause except where data-name-2 is the subject of the entry.

7. There must not be any entry that contains an OCCURS clause between the data-names in the KEY IS phrase and the subject of the entry, except where data-name-2 is the subject of the entry.

8. All data-names used in the OCCURS clause may be qualified; however, they must not be subscripted or indexed.

9. An INDEXED BY phrase is required if the subject of this entry, or an entry subordinate to this entry, is to be referenced by indexing. The index-names identified by this phrase are not defined elsewhere; not representing data, the index-names cannot be associated with any data record or be referred to in a USING phrase.

10. Each index-name must be unique within the program.

11. Where both integer-1 and integer-2 are used, the value of integer-1 must be less than the value of integer-2.

12. The data description of data-name-1 must describe a positive integer.

13. A data-description-entry that contains Format 2 of the OCCURS clause may be followed, within that record description, only by data description entries that are subordinate to it.

14. In Format 2, the data item defined by data-name-1 must not fall within the range of the first character position defined by the data-description-entry containing the OCCURS clause and the last character position defined by the record-description-entry containing that OCCURS clause.

15. If data-name-2 is not the subject of the OCCURS entry, then:

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

    - Items identified by the data-name in the KEY IS phrase must not contain an OCCURS clause.

- There must not be any entry that contains an OCCURS clause between the items identified by the data-names in the KEY IS phrase and the subject of this entry.

General Rules

1. The OCCURS clause defines tables or homogeneous sets of repeated data items. When the OCCURS clause is used, the data-name which is the subject of the entry, and any items subordinate to it, must be referred to by subscripting or indexing except in the SEARCH verb. An example is given with Rule 6 below.

2. Except for the OCCURS clause itself, all data description clauses associated with an item containing an OCCURS clause apply to each occurrence of the item being described.

3. In Format 2, the current value of data-name-1 represents the number of occurrences.

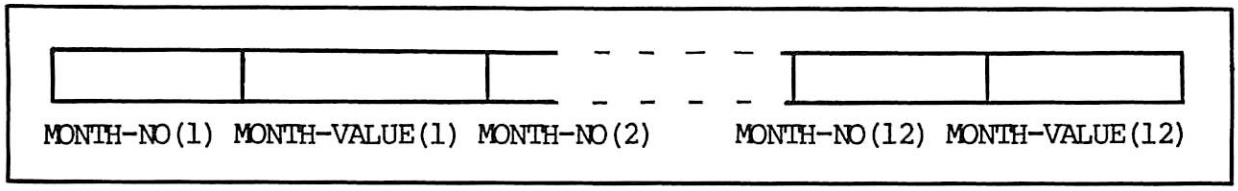
4. The KEY IS phrase indicates that the repeated data is arranged in ascending or descending order according to the values contained in data-name-2, data-name-3, etc. The ascending or descending order is determined by the rules for the comparison of operands in Chapter 4. The data-names are listed in their descending order of significance.

5. When the INDEXED BY phrase is omitted, subscripting is used to indicate an individual element within a list, or within a table of like elements that do not have individual data-names.

6. When the INDEXED BY phrase is used, an index is assigned to a table of like elements, with individual items in the table being identified by index-name. For example, the following code defines a table MONTH-TAB of 12 items, indexed by INDX.

```
      .
      .
      .
05  MONTH-TAB OCCURS 12 TIMES
    ASCENDING KEY MONTH-NO
    INDEXED BY INDX.
    10  MONTH-NO     PIC 99.
    10  MONTH-VALUE  PIC XXX.
      .
      .
      .
```

This code creates a storage area that may be represented as in Figure 10-1.



A 12-element Table
Figure 10-1

References to an individual item in the table are made by means of an index or subscript. Thus, three ways of referring to the subgroup MONTH-NO of the fourth element of MONTH-TAB, assuming INDX and DATA-NM both have the value 4, are:

```
MONTH-NO(4)
MONTH-NO(INDX)
MONTH-NO(DATA-NM)
```

Subscripting of a table whose definition includes INDEXED BY is also allowed.

7. The number of occurrences of the subject entry is defined as follows:

   - If DEPENDING is not used, the value of integer-2 represents the exact number of occurrences.
   - If DEPENDING is used, the current value of the data item referenced by data-name-1 represents the number of occurrences.

   This clause specifies that the subject of this entry has a variable number of occurrences. The value of integer-2 represents the maximum number of occurrences and the value of integer-1 represents the minimum number of occurrences. This does not imply that the length of the subject of the entry is variable, but that the number of occurrences is variable.

   The value of data-name-1 must fall within the range integer-1 through integer-2. Reducing the value of data-name-1 makes the contents of data items whose occurrence numbers now exceed the value of data-name-1 unpredictable.

USAGE

The USAGE clause specifies the format of an index data item in computer storage.

Format

[USAGE IS] INDEX

Syntax Rules

1. An elementary item described with the USAGE IS INDEX clause is called an index data item. An index data item can be referenced explicitly only in a SEARCH or SET statement, a relation condition, the USING phrase of a PROCEDURE division header, or the USING phrase of a CALL statement.

2. The SYNCHRONIZED, JUSTIFIED, PICTURE, VALUE, and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

General Rules

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

2. An index data item contains a value that must correspond to an occurrence number of a table element. The elementary item cannot be a conditional variable. If a group item is described with the USAGE IS INDEX clause, the elementary items in the group are all index data items. The group itself is not an index data item and cannot be used in the SEARCH or SET statement or in a relation condition.

3. An index data item can be part of a group that is referred to in a MOVE or input-output statement, in which case no conversion will take place.

4. The format of the index data item is described in the section DATA REPRESENTATION AND ALIGNMENT in Chapter 4.

5. The maximum value of an index item or index data item is listed in Appendix J.

INDEXED BY

The INDEXED BY phrase appears in the OCCURS clause format. The INDEXED BY phrase defines the index name to be used with the subject of the OCCURS clause, and thus allows that subject to be referred to with indexing. The index-name identified by this phrase is not defined elsewhere; an index-name is declared not by the usual method of level-number, name, and data description clauses, but implicitly by appearance in the "INDEXED BY index-name" phrase of the OCCURS clause.

Format

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

General Rules

1. An index-name must be uniquely named.

2. An index-name may be modified only by the SET verb, the SEARCH verb, and the PERFORM verb.

3. A maximum of eight indexes may be used in an indexed reference.

4. The format of an index-name item is described in the section DATA REPRESENTATION AND ALIGNMENT in Chapter 4. Its maximum value is listed in Appendix J.

PROCEDURE DIVISION

SET

The SET statement permits the manipulation of index-names and index data items, for table-handling purposes.

Format 1

```
SET { index-name-1 [, index-name-2] ... } TO { index-name-3 }
    { data-name-1 [, data-name-2] ...  }     { data-name-3 }
                                             { integer-1   }
                                             { arith-expr-1 }
```

Format 2

```
SET index-name-4 [, index-name-5] ... { UP BY   } { data-name-4 }
                                      { DOWN BY } { integer-2   }
                                                  { arith-expr-2 }
```

Syntax Rules

1. All references to index-name-1, data-name-1, and index-name-4 apply equally to index-name-2, data-name-2, and index-name-5, respectively.

2. The data-name-4 must be described as an elementary numeric integer.

3. The data-names 1 and 3 must name either index data items or elementary integer items.

4. The integers 1 and 2 may be signed. However, integer-1 must have a positive value. Arithmetic-expression-1 and arithmetic-expression-2 must evaluate to positive integers.

5. The use of arithmetic expressions in the SET statement is a Prime extension.

General Rules

1. Index-names are related to a specific table and are defined with the INDEXED BY clause.

2. If index-name-3 is specified, the value of the index before the execution of the SET statement must not exceed the maximum number of elements in the associated table.

3. In Format 1, the following action occurs:

   - The index-name-1 is set to a value causing it to refer to the table element that corresponds in occurrence number to the value of the name after TO. If data-name-3 is an index data item, or if index-name-3 is related to the same table as index-name-1, no conversion takes place.

   - If data-name-1 is an index data item, it may be set equal to the contents of either index-name-3 or data-name-3, where data-name-3 is also an index data item; no conversion takes place in either case.

   - If data-name-1 is not an index data item, it may be set only to an occurrence number which corresponds to the value of index-name-3. Neither data-name-3 nor integer-1 can be used in this case.

   - The process is repeated for index-name-2, data-name-2, and so on. Each time, the value of index-name-3 or data-name-3 is used as it was at the beginning of the execution of the statement.

4. In Format 2, the index name or names following SET are incremented or decremented by the value after UP or DOWN, respectively. Each time the value of data-name-4 is used as it was at the beginning of the execution of the statement.

5. Table 10-1 represents the validity of various operand combinations in the SET statement.

Table 10-1
Validity of Operand Combinations in the SET Statement

Sending Item	Type of Receiving Item		
	Integer Data Item	Index-name	Index Data Item
Integer arith-expr		Valid	
Integer literal		Valid	
Integer data item		Valid	
Index-name	Valid	Valid	Valid*
Index data item		Valid*	Valid*

* No conversion takes place

SEARCH

The SEARCH statement is used to search a table for an element that satisfies the specified condition. The associated index-name is adjusted 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         }

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

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   } }
        {                           } { arith-expr-1} }
        { condition-name-1                            }

  [ AND { data-name-4 { IS EQUAL TO } { data-name-5 } } ] ...
        {             { IS =        } { literal-2   } }
        {                           } { arith-expr-2} }
        { condition-name-2                            } ]

  { imperative-statement-2 }
  { NEXT SENTENCE          }
```

Syntax Rules

1. In both Formats 1 and 2, data-name-1 must not be subscripted or indexed, but its description must contain an OCCURS clause and an INDEXED BY clause. The description of data-name-1 in Format 2 must also contain the KEY IS phrase in its OCCURS clause, so it must be an ordered table.

2. Data-name-2, when specified, must be described as USAGE IS INDEX, or as a numeric elementary data item without any positions to the right of the assumed decimal point.

3. In Format 1 or condition-1, condition-2 may be any condition as described under CONDITIONAL EXPRESSIONS in Chapter 4.

4. In Format 2, all referenced condition-names must be defined as having only a single value. The data-name associated with a condition-name must appear in the KEY clause of data-name-1. Each data-name-2 or data-name-4 may be qualified. Further, each data-name-2 or data-name-4 must be indexed by the first index-name associated with data-name-1 along with other indexes or literals as required.

5. In Format 2, when a data-name in the KEY clause of data-name-1 is referenced, or when a condition-name associated with a data-name in the KEY clause of data-name-1 is referenced, all preceding data-names in the KEY clause of data-name-1 or their associated condition-names must also be referenced.

General Rules

1. The Format-1 SEARCH statement enables a serial search operation, starting with the current index setting.

   - If, at the start of execution of the SEARCH statement, the index-name associated with data-name-1 contains a value greater than the highest permissible occurrence number for data-name-1, the specified imperative-statement-1 is executed. If the AT END phrase is not specified, control passes to the next executable sentence.

   - If, at the start of execution of the SEARCH statement, the index-name associated with data-name-1 contains a value not greater than the highest permissible occurrence number for data-name-1, the SEARCH statement operates by evaluating the conditions in the order in which they are written, making use of the index settings, wherever specified, to determine the occurrence of those items to be tested. If none of the conditions are satisfied, the index-name for data-name-1 is incremented to obtain reference to the next

occurrence. The process is repeated, using the new index-name settings. If the new value of the index-name settings for data-name-1 corresponds to a table element outside the permissible range of occurrence values, the search terminates as indicated in the paragraph above. If one of the conditions is satisfied upon its evaluation, the search terminates immediately and the imperative statement associated with that condition is executed; the index-name remains set at the occurrence which caused the condition to be satisfied.

Example

The following code assumes the table MONTH-TAB has been defined as shown in the example with OCCURS. The SEARCH statement causes a search of MONTH-TAB, changing the value of INDX until the element whose position is specified by INDX has the value of MONTH-ACCEPT.

```
05  MONTH-TAB OCCURS 12 TIMES INDEXED BY INDX
    ASCENDING KEY MONTH-NO.
    10  MONTH-NO      PIC 99.
    10  MONTH-VALUE   PIC XXX.
          .
          .
          .

FIND-MONTH.
   SEARCH MONTH-TAB
      WHEN MONTH-NO(INDX) = MONTH-ACCEPT
      MOVE MONTH-VALUE(INDX)  TO PRINT-MONTH.
```

2. In Format 1, if the VARYING phrase is not used, the index-name that is used for the search operation is the first (or only) index-name appearing in the INDEXED BY phrase of data-name-1. Any other index-names for data-name-1 remain unchanged.

3. In Format 1, if the VARYING index-name-1 phrase is specified, and if index-name-1 appears in the INDEXED BY phrase of data-name-1, that index-name is used for this search. If this is not the case, or if the VARYING data-name-2 phrase is specified, the first (or only) index-name given in the INDEXED BY phrase of data-name-1 is used for the search. In addition, the following operations will occur:

   - If the VARYING index-name-1 phrase is used, and if index-name-1 appears in the INDEXED BY phrase of another table entry, index-name-1 is incremented by the same amount as, and at the same time as, the index-name associated with data-name-1 is incremented.

- If the VARYING data-name-2 phrase is specified, and data-name-2 is an index data item, then data-name-2 is incremented by the same amount as, and at the same time as, the index associated with data-name-1 is incremented. If data-name-2 is not an index data item, data-name-2 is incremented by the value 1 at the same time as the index-name associated with data-name-1.

4. In a Format-2 SEARCH statement, the results of the SEARCH ALL operation are predictable only when:

   - The data in the table is ordered in the same manner as described in the ASCENDING/DESCENDING KEY clause associated with the description of data-name-1.

   - The contents of the key(s) referenced in the WHEN clause are sufficient to identify a unique table element.

5. When a Format-2 SEARCH ALL is used, the initial setting of the index-name for data-name-1 is ignored and its setting is varied during the search operation. However, at no time is the index-name set to a value that exceeds the number of elements in the table, or that is less than the value that corresponds to the first element of the table. The length of the table is discussed in the OCCURS clause at the beginning of this chapter.

   If any of the conditions specified in the WHEN clause cannot be satisfied for any setting of the index within the permitted range, control is passed to imperative-statement-1 of the AT END phrase, when specified, or to the next executable sentence. In either case, the final setting of the index is not predictable. If all the conditions can be satisfied, the index indicates an occurrence that allows the conditions to be satisfied, and control passes to imperative-statement-2.

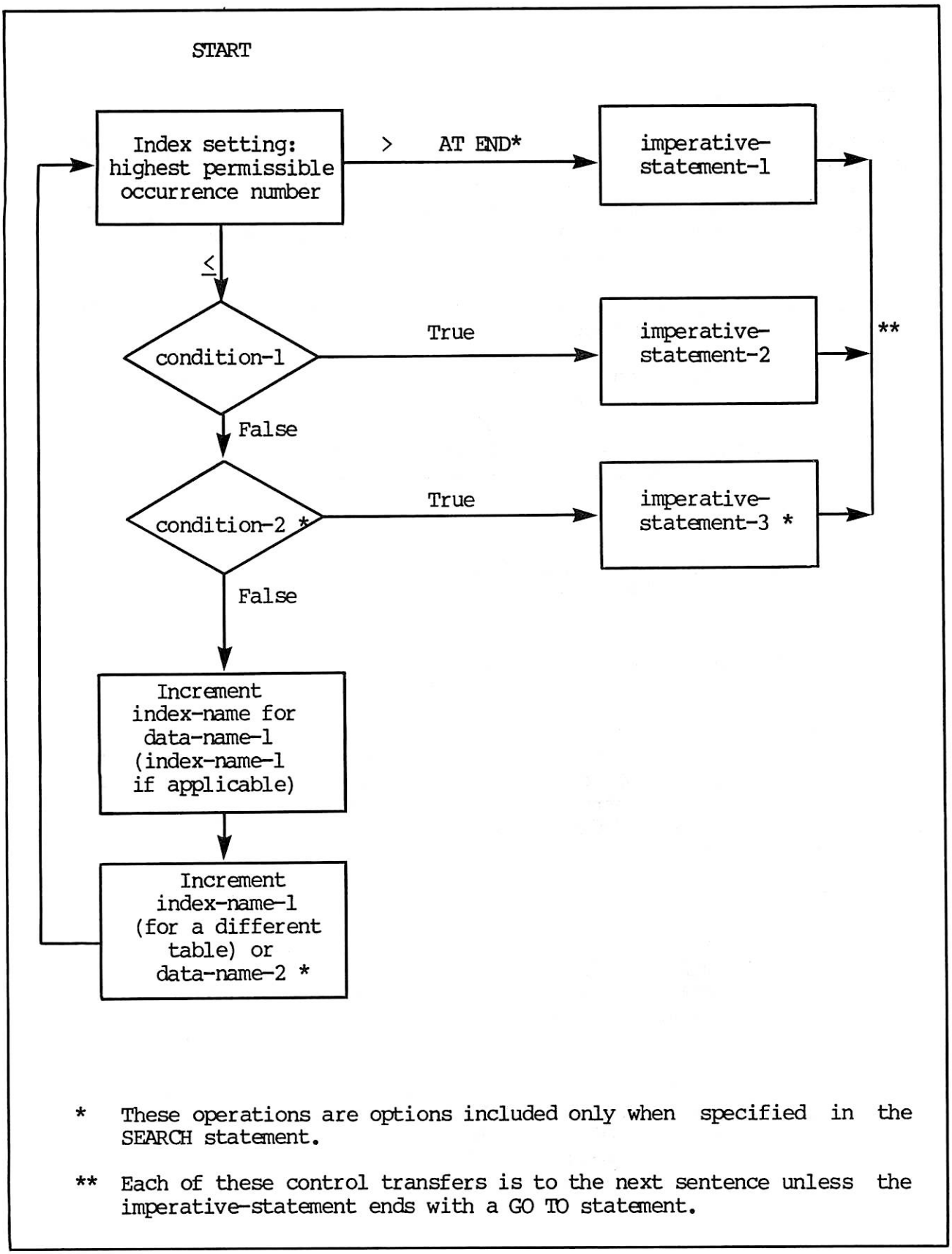
6. If imperative-statement-1, imperative-statement-2, or imperative-statement-3 does not terminate with a GO TO statement, control passes to the next sentence.

7. In Format 2, the index-name that is used for the search operation is the first (or only) index-name that appears in the INDEXED BY clause of data-name-1. Any other index-names for data-name-1 remain unchanged.

8. If data-name-1 is a data item subordinate to another data item containing an OCCURS clause (providing for a two- or three-dimensional table), an index-name must be associated with each dimension of the table. This is accomplished through the INDEXED BY phrase of the OCCURS clause. Only the setting of the index-name associated with data-name-1 (and data-name-2 or index-name-1, if present) is modified by the execution of the SEARCH statement.

To search an entire two- or three-dimensional table, it is necessary to execute a SEARCH statement several times. Prior to each execution of a SEARCH statement, SET statements must be executed to adjust index-names to appropriate settings.

9. A flowchart of the Format-1 SEARCH operation containing two WHEN phrases is presented in Figure 10-2.



* These operations are options included only when specified in the SEARCH statement.

** Each of these control transfers is to the next sentence unless the imperative-statement ends with a GO TO statement.

Format-1 SEARCH Flowchart
Figure 10-2

STRATEGY

Table Initialization

Table initialization, if required, may be achieved either in the WORKING-STORAGE section or in the PROCEDURE division. The VALUE clause is not permitted in a data-description-entry specifying an OCCURS or REDEFINES clause, or in any entry subordinate to one specifying an OCCURS or REDEFINES clause. The following paragraphs suggest means of assigning values to table elements.

In the WORKING-STORAGE section of the DATA division, tables can be initialized in one of two ways:

- If the elements in a table do not need to be individually initialized, then the VALUE clause may be specified in the data-description-entry containing the table name. The subordinate data-description-entry should then be given an OCCURS clause defining the structure of the table.

  Examples:

```
01  A-TABLE       VALUE ZEROS.
  05  B-TABLE     PIC 9(3)  OCCURS 100 TIMES.


01  STATE-TABLE   VALUE 'CALAMAPAVA'.
  05  STATE       PIC XX  OCCURS 5 TIMES.
```

- If the elements in a table need to be individually initialized, the table must first be defined as a nontable structure with the desired number of characters. A VALUE clause can be specified in each element entry of the nontable structure. The structure can then be redefined as a table with REDEFINES plus a subordinate entry containing an OCCURS clause.

  Example:

```
01  WAREHOUSE.
     05  FILLER     PIC 99       VALUE 10.
     05  NAME       PIC X(22)    VALUE 'BOSTON DISTRICT BRANCH'.
     05  FILLER     PIC 99       VALUE 11.
     05  FILLER     PIC X(22)    VALUE 'NEW YORK CITY BRANCH  '.
     05  FILLER     PIC 99       VALUE 12.
     05  FILLER     PIC X(22)    VALUE 'HOUSTON HOME OFFICE   '.
01  WARE-HOUSE REDEFINES WAREHOUSE.
     05  HOUSES   OCCURS 3 TIMES.
          10  HOUSE-NO      PIC 99.
          10  HOUSE-NAME    PIC X(22).
```

In the PROCEDURE division, a table can be initialized with MOVE statements:

```
MOVE '10BOSTON DISTRICT BRANCH11NEW YORK CITY BRANCH  12HOUST
-    'ON HOME OFFICE    ' TO WAREHOUSE.
```

Indexing and Subscripting

Indexing and subscripting are the two methods of accessing the individual elements in a table established by the OCCURS clause. An index is an index-name in an INDEXED BY phrase in an OCCURS clause. A subscript is an integer or a data-name. To specify a desired table element, follow the table element's data-name by a parenthesized index or subscript, such as HOUSE-NO (3). The value of an index or subscript should represent the occurrence number of the desired element.

Indexing: The general format for direct indexing and relative indexing is:

```
{ data-name      } ( { index-name-1 [ { } literal-2      ] }
{ condition-name }   { literal-1    [ { ± } arith-expr-2 ] }
                     { arith-expr-1 [ { }                ] }

  [ , { index-name-2 [ { } literal-4      ] } ]
      { literal-3    [ { ± } arith-expr-4 ] }
      { arith-expr-3 [ { }               ] }

  [ , { index-name-3 [ { } literal-6      ] } ] ] )
      { literal-5    [ { ± } arith-expr-6 ] }
      { arith-expr-5 [ { }               ] }
```

When a statement that refers to an indexed table element is executed, the value in the associated index must be neither less than 1, nor greater than the highest occurrence number of an element in the table. This restriction applies equally to direct indexing and relative indexing.

If an arithmetic expression is part of an indexed reference, it must evaluate to an integer at the time of reference.

Direct Indexing: Direct indexing is specified by using an index-name alone within parentheses, for example, ELEMENT(INDX1).

Consider the following illustration:

```
        .
        .
        .
01 TABLE-A.
    05 ELEMENT OCCURS 6 TIMES INDEXED BY INDX1.
        .
        .
        .
    SET INDX1 TO 4.
    MOVE ELEMENT (INDX1) TO PRINT-FIELD.
        .
        .
        .
```

ELEMENT(INDX1) in the example above refers to the fourth element of the table. The MOVE statement moves the contents of the fourth occurrence of ELEMENT to a field called PRINT-FIELD.

Relative Indexing: Relative indexing uses an arithmetic expression to compute the location of a table element. Relative indexing may be used wherever indexing can be used. Using the sample TABLE-A defined in the example above, the same results could be achieved with relative indexing. If INDX1 has a value of 1, the fourth element of TABLE-A can be moved to PRINT-FIELD with this statement:

```
MOVE ELEMENT(INDX1 + 3) TO PRINT-FIELD.
```

In relative indexing, index-name is followed by a space, followed by one of the operators + or -, followed by another space, followed by an unsigned integer numeric literal or arithmetic expression, all delimited by the balanced pair of separators left parenthesis and right parenthesis.

The occurrence number resulting from relative indexing is determined by incrementing or decrementing the index by the value of the literal or arithmetic expression.

Subscripting: Subscripting may be used in lieu of indexing.

The format for subscripting is:

```
{ data-name      }
{ condition-name } (subscript-1 [, subscript-2 [, subscript-n] ··· ])
```

A subscript must be delimited by a pair of parentheses following the table element data-name. When two or more subscripts are required, they are written in the order of successively less inclusive dimensions of the data organization, and may be separated by commas. A maximum of eight levels of subscripting is permitted for any given data item.

A subscript value is changed in the PROCEDURE division via the MOVE, SEARCH, PERFORM, ADD, SUBTRACT, MULTIPLY, DIVIDE, or COMPUTE verbs.

The subscript can be represented by a positive integer literal, a data-name, or an integer arithmetic expression. The data-name must be a numeric elementary item that represents an integer. Further, the data-name as subscript may be qualified or subscripted.

The subscript data-name may be signed, but its value must be positive. The subscript value indicates the position of the item in a table. The lowest value permitted is 1, indicating the first position in the table. Subsequent positions are indicated by sequential values 2, 3, 4, and so on, up to the highest permissible value, which is the maximum number of occurrences of the item specified in the OCCURS clause.

Three kinds of subscripting can be used on any table and data-name subscripting.

Literal Subscripting: An integer in parentheses is used. For example, consider this three-element array:

```
01 ARRAY.
   05 ELEMENT, OCCURS 3, PICTURE S9(4), SIGN TRAILING SEPARATE.
```

The statement below results in the contents of the second ELEMENT of ARRAY being moved to the field called PART-NO.

```
MOVE ELEMENT(2) TO PART-NO.
```

Data-name Subscripting: An additional data-description-entry is required to define a data-name to be used as subscript (SUBSCRIPTNO in this example):

```
01  ARRAY.
    05  ELEMENT, OCCURS 3, PICTURE X(4).
    .
    .
    .
01  SUBSCRIPTNO            PIC 99.
01  PART-NO                PIC X(4).
    .
    .
    .
    MOVE 2 TO SUBSCRIPTNO.
    PERFORM 050-TABLERUN.
```

```
      .
      .
      .
050-TABLERUN.
    MOVE ELEMENT(SUBSCRIPTNO) TO PART-NO.
```

Subscripting with Arithmetic Expressions -- Prime Extension: This form of subscripting is similar to data-name subscripting except that any subscript may be an arithmetic expression. The arithmetic expression, at the time of reference, should evaluate to a positive integer.

Multidimensional Tables

When a table has more than one dimension, the data-name of the desired item is followed by a list of subscripts, one for each OCCURS clause to which the item is subordinate.

In such a list, the first subscript applies to the first OCCURS clause to which the item is subordinate. The second subscript applies to the next most encompassing level. The third subscript applies to the next lower level OCCURS clause being accessed, and so on.

The following example presents DATA division entries for a multidimensional table, TABLE-PLUS.

```
01 TABLE-PLUS.
   05 TYPE OCCURS 10 TIMES.
      10 PART-NO          PIC X(4).
      10 COLOR            PIC X OCCURS 10 TIMES.
      10 CONTROL OCCURS 7 TIMES.
         15 C1            PIC X.
         15 C2            PIC XX OCCURS 4 TIMES.
```

The statement:

```
MOVE C2(8, 6, 4) TO TEMP.
```

would move the contents of the fourth occurrence of the field C2, in the sixth occurrence of the field CONTROL, in the eighth occurrence of the field TYPE, to a field called TEMP.

Similarly, the statement:

```
MOVE C2(10, 7, 4) TO TEMP.
```

would move the contents of the last occurrence of the field C2 to the field labeled TEMP.

EXAMPLE

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.                     BUDGET.
 AUTHOR.                         PRIMATE1.
 INSTALLATION.                   PRIME.
 DATE-COMPILED.
*****************************************************************
 REMARKS. THE PROGRAM READS BUDGET LIMITS AND A TRANSACTION FILE.
     IF ANY BUDGET LIMIT ENTRIES ARE MISSING, ZERO AMOUNTS ARE
     ASSUMED.  THE TRANSACTION FILE MUST BE SORTED
     BY  ACCOUNT WITHIN DEPARTMENT.

     THE PROGRAM MAKES A TABLE FOR EACH CATEGORY FOR EACH MONTH,
     INCLUDING BUDGET LIMIT AND YEAR-TO-DATE TOTALS SPENT.

     THE PROGRAM PRODUCES ONE REPORT.  FOR EACH ACCOUNT NUMBER,
     IT SEARCHES ALL CATEGORIES, COMPARES EXPENDITURES
     WITH BUDGETED LIMITS, AND PRINTS ANY CATEGORIES THAT ARE
     OVER THE BUDGETED LIMIT.
*****************************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. PRIME.
 OBJECT-COMPUTER. PRIME.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT INPUT-FILE ASSIGN TO PFMS.
     SELECT PRINT-FILE ASSIGN TO PRINTER.
*****************************************************************
 DATA DIVISION.
 FILE SECTION.
 FD  INPUT-FILE COMPRESSED,
     VALUE OF FILE-ID IS 'BUDGET.DATA',
     LABEL RECORDS ARE STANDARD.
 01  ENTRY                      PIC X(80).
*
 FD  PRINT-FILE,
     LABEL RECORDS ARE OMITTED.
 01  PRINT-LINE                 PIC X(132).
*
 WORKING-STORAGE SECTION.
 77  ACCT-SAVE-WS               PIC X(3).
 77  EXCESS-COUNT-WS            PIC S999 COMP-3    VALUE 0.
 77  NO-MORE-INPUT              PIC X              VALUE 'N'.
*****************************************************************
*TWO-DIMENSIONAL TABLE FOR DEPTS AND CATEGORIES, INDEXED.       *
*****************************************************************
 01  SELECTED-TABLE.
     05 DEPTS                   PIC X(1120) VALUE SPACES.
     05 DEPTS-SUB REDEFINES DEPTS.
        10 THEDEPT OCCURS 8 TIMES,  INDEXED BY DEPT-SUB.
         13 THEREST OCCURS 10, INDEXED BY CAT-SUB.
           15  BUDGETED-AMT     PIC 9(5)V99.
```

```
           15  AMT-SPENT          PIC 9(5)V99.
*************************************************************
* ONE-DIMENSIONAL TABLE FOR CATEGORY NAMES IN EXCESS-LINE *
*************************************************************
 01  NAMES1.
     05 NAME-TABLE                  PIC X(100) VALUE 'AUTO      CLOT
-    'HING  FOOD      INSURANCE MAINTENANCMEDICAL   MORTGAGE  REC
-    'REATIONUTILITIES MISC    '.
     05 CATEGORY-NAME1 REDEFINES NAME-TABLE.
        10 CATEGORY-NAME OCCURS 10 TIMES,
         INDEXED BY NAME-SUB       PIC X(10).
*
*****************************************************************
*    WORK LINES                                                 *
*****************************************************************
*LIMIT-LINE IS BUDGET LIMIT:
 01  LIMIT-LINE.
     05 CODE-LT                     PIC X.
     05 ACCT-LT                     PIC X(3).
     05 CAT-LT                      PIC 99.
     05 DATE-LT.
        10 MONTH-LT                 PIC 99.
        10 DAY-YR                   PIC 9(4).
     05 AMT-LT                      PIC 9(5)V99.
*DATA-NAME FOR SORTED RECORDS:
 01  INPUT1.
     05 CODE-IN                     PIC X.
     05 ACCT-IN                     PIC X(3).
     05 CAT-IN                      PIC 99.
     05 DATE-IN.
        10 MONTH-IN                 PIC 99.
        10 DAY-IN                   PIC 99.
        10 YEAR-IN                  PIC 99.
     05 AMOUNT-IN                   PIC 9(5)V99.
     05 FILLER                      PIC X(61).
*
 01  EXCESS-WORK.
     05 CATEGORY                    PIC X(10)     VALUE SPACES.
     05 AMOUNT                      PIC 9(10)V99  VALUE 0 COMP-3.
     05 BUDGET-LIMIT                PIC 9(10)V99  VALUE 0 COMP-3.
     05 OVER                        PIC 9(10)V99  VALUE 0 COMP-3.
     05 PERCENT                     PIC 999V99    VALUE 0 COMP-3.
*
*****************************************************************
*                PRINT LINES                                    *
*****************************************************************
 01  HEADING1.
     05 CTL1                        PIC X     VALUE '1'.
     05 FILLER                      PIC X(11) VALUE 'ACCOUNT    '.
     05 ACCT-PRINT                     PIC X(3)  VALUE SPACES.
     05 FILLER                      PIC X(115)
       VALUE '              CATEGORIES EXCEEDING BUDGET
-    '                                                  '.
```

```
 01  HEADING2.
     05 CTL2                      PIC X     VALUE '0'.
     05 FILLER                    PIC X(71)
        VALUE 'CATEGORY        YTDSPENT       BDGTD AMT    AMOUNT
-    'OVER                '.
 01  EXCESS-PRINT.
     05 CTRL-EXC                  PIC X     VALUE ' '.
     05 CATEGORY                  PIC X(10) VALUE SPACES.
     05 AMOUNT                    PIC Z(9)9.99.
     05 FILLER                    PIC X     VALUE SPACES.
     05 BUDGET-LIMIT              PIC Z(9)9.99.
     05 FILLER                    PIC XX   VALUE SPACES.
     05 OVER                      PIC Z(9)9.99.
 01  MESSAGE-LINE                 PIC X(49)
       VALUE '1NO CATEGORIES HAVE AN AMOUNT THAT EXCEEDS BUDGET'.
*****************************************************************
*
 PROCEDURE DIVISION.
*
 DECLARATIVES.
   INPUT-ERROR SECTION. USE AFTER ERROR PROCEDURE ON INPUT-FILE.
   FIRST-PARAGRAPH.
     DISPLAY '*** I-O ERROR ON INPUT FILE:  ***'.
     STOP RUN.
 END DECLARATIVES.
*
 000-MAINLINE.
     PERFORM 015-ZERO-TABLES.
     PERFORM 017-OPEN.
     PERFORM 020-BUDGETED-TOTALS UNTIL CODE-LT = 'T'.
     PERFORM 030-PROCESS-TRANS.
     CLOSE PRINT-FILE, INPUT-FILE.
     STOP RUN.
*
 017-OPEN.
     OPEN INPUT INPUT-FILE,
          OUTPUT PRINT-FILE.
     MOVE SPACES TO PRINT-LINE.
     WRITE PRINT-LINE AFTER ADVANCING PAGE.
     READ INPUT-FILE INTO LIMIT-LINE,
         AT END DISPLAY 'EMPTY FILE',
      CLOSE INPUT-FILE, PRINT-FILE
      STOP RUN.
*
 015-ZERO-TABLES.
     SET DEPT-SUB, CAT-SUB TO 1.
     PERFORM 018-ZERO UNTIL DEPT-SUB>8.
 018-ZERO.
     PERFORM 019-ZERO UNTIL CAT-SUB > 10.
     SET DEPT-SUB UP BY 1.
 019-ZERO.
     MOVE ZEROS TO BUDGETED-AMT(DEPT-SUB, CAT-SUB).
     MOVE ZEROS TO AMT-SPENT(DEPT-SUB, CAT-SUB).
     SET CAT-SUB UP BY 1.
```

```
*
 020-BUDGETED-TOTALS.
*************************************************************
*  READ BUDGET LIMITS INTO TABLE BY MONTHS.                 *
*  USE DEPT-NO AND CAT-NO ON LIMIT-CARD AS INDEXES:         *
*************************************************************
     SET  CAT-SUB TO CAT-LT.
     IF ACCT-LT = 'ABC' SET DEPT-SUB TO 1
         ELSE IF ACCT-LT = 'DEF' SET DEPT-SUB TO 2
         ELSE IF ACCT-LT = 'GHI' SET DEPT-SUB TO 3
         ELSE IF ACCT-LT = 'JES' SET DEPT-SUB TO 4
         ELSE IF ACCT-LT = 'XYZ' SET DEPT-SUB TO 7
         ELSE SET DEPT-SUB TO 8.
     MOVE AMT-LT TO BUDGETED-AMT(DEPT-SUB, CAT-SUB).
     READ INPUT-FILE INTO LIMIT-LINE,
         AT END MOVE 'T' TO CODE-LT,
                DISPLAY 'NO TRANS CARDS'.
*
 030-PROCESS-TRANS.
     READ INPUT-FILE INTO INPUT1,
         AT END DISPLAY 'EMPTY FILE',
         CLOSE INPUT-FILE, PRINT-FILE
         STOP RUN.
     MOVE ACCT-IN TO ACCT-SAVE-WS.
     PERFORM 031-PROCESS-REST
         UNTIL NO-MORE-INPUT = 'Y'.
*
 031-PROCESS-REST.
     PERFORM 033-CHECK-FOR-ERRORS.
     PERFORM 035-CONTROL-BREAK-CHECK.
     PERFORM 036-MAKE-TABLES.
     PERFORM 037-READ-NEXT.
*
 033-CHECK-FOR-ERRORS.
*     NOT INCLUDED
*
 035-CONTROL-BREAK-CHECK.
     IF ACCT-IN NOT EQUAL ACCT-SAVE-WS,
         PERFORM 040-NEXT-ACCT.
*
 036-MAKE-TABLES.
*********************************************************
* SET DEPT-SUB, CAT-SUB FOR TABLE, ADD TO THOSE TOTALS.  *
*********************************************************
     SET CAT-SUB TO CAT-IN.
     IF ACCT-IN = 'ABC' SET DEPT-SUB TO 1
         ELSE IF ACCT-IN = 'DEF' SET DEPT-SUB TO 2
         ELSE IF ACCT-IN = 'GHI' SET DEPT-SUB TO 3
         ELSE IF ACCT-IN = 'JES' SET DEPT-SUB TO 4
         ELSE SET DEPT-SUB TO 8.
     ADD AMOUNT-IN TO AMT-SPENT(DEPT-SUB, CAT-SUB).
*
 037-READ-NEXT.
     READ INPUT-FILE INTO INPUT1,
```

```
            AT END MOVE 'Y' TO NO-MORE-INPUT,
                PERFORM 040-NEXT-ACCT.
*
 040-NEXT-ACCT.
     MOVE ACCT-SAVE-WS TO ACCT-PRINT.
     MOVE ACCT-IN TO ACCT-SAVE-WS.
     MOVE ZEROS TO EXCESS-COUNT-WS.
     MOVE SPACES TO PRINT-LINE.
     WRITE PRINT-LINE FROM HEADING1 AFTER ADVANCING 4.
     WRITE PRINT-LINE FROM HEADING2 AFTER ADVANCING 2.
     SET CAT-SUB TO 1.
     PERFORM 041-SEARCH-FOR-EXCESS UNTIL CAT-SUB > 10.
     IF EXCESS-COUNT-WS = 0, PERFORM 046-MESSAGE.
     MOVE 0 TO EXCESS-COUNT-WS.
*
 041-SEARCH-FOR-EXCESS.
*************************************************************
* LINEAR SEARCH OF ONE DEPARTMENT BY CATEGORIES:            *
*************************************************************
     IF  AMT-SPENT(DEPT-SUB, CAT-SUB) > BUDGETED-AMT
             (DEPT-SUB, CAT-SUB),
         ADD 1 TO EXCESS-COUNT-WS,
         PERFORM 045-PRINT-EXCESS.
     SET CAT-SUB UP BY 1.
*
 045-PRINT-EXCESS.
     SET NAME-SUB TO CAT-SUB.
     MOVE CATEGORY-NAME(NAME-SUB) TO CATEGORY OF EXCESS-WORK.
     MOVE AMT-SPENT(DEPT-SUB, CAT-SUB) TO AMOUNT OF EXCESS-WORK.
     MOVE BUDGETED-AMT(DEPT-SUB, CAT-SUB) TO
         BUDGET-LIMIT OF EXCESS-WORK.
*     SUBTRACT BUDGETED-AMT(DEPT-SUB, CAT-SUB)
*       FROM AMT-SPENT(DEPT-SUB, CAT-SUB)
*         GIVING OVER OF EXCESS-WORK.
*    MOVE CORR EXCESS-WORK TO EXCESS-PRINT.
     COMPUTE OVER OF EXCESS-WORK = AMT-SPENT(DEPT-SUB, CAT-SUB)
         - BUDGETED-AMT(DEPT-SUB, CAT-SUB).
     MOVE CORR EXCESS-WORK TO EXCESS-PRINT.
     MOVE SPACES TO PRINT-LINE.
     WRITE PRINT-LINE FROM EXCESS-PRINT.
 046-MESSAGE.
     MOVE SPACES TO PRINT-LINE.
     WRITE PRINT-LINE FROM MESSAGE-LINE.
```

This program, stored as BUDGET.TABLE.CBL, may be compiled, loaded, and executed with the following dialog. Sample input and output is given below.

```
OK, CBL BUDGET.TABLE -L

[CBL rev X.X]
OK, SEG -LOAD
[SEG rev X.X]
$ LO BUDGET.TABLE
$ LI CBLLIB
$ LI
LOAD COMPLETE
$ EXEC
OK,
```

Input File (BUDGET.DATA)

```
OK, SLIST BUDGET.DATA
BABC010132780300200
BABC020232780346200
BABC030332780020000
BABC040432780000500
BABC050532780200060
BDEF080532780200000
BGHI080332780098300
BGHI090432780090000
TABC031025789998930
TABC031025780000116
TABC051021780000984
TABC021004780000386
TABC031004780008512
TABC031004780004000
TABC011001780030000
TABC041001780001000
TABC030802780008651
TABC030730780000450
TABC030725780008015
TDEF080720780000430
TGHI090615780025600
TGHI080614780003050
OK,
```

Output File (PRINT-FILE)

```
ACCOUNT    ABC                CATEGORIES EXCEEDING BUDGET

CATEGORY          YTDSPENT       BDGTD AMT     AMOUNTOVER
FOOD                297.44          200.00          97.44
INSURANCE            10.00            5.00           5.00
```

```
ACCOUNT    DEF              CATEGORIES EXCEEDING BUDGET

CATEGORY         YTDSPENT        BDGTD AMT     AMOUNTOVER
NO CATEGORIES HAVE AN AMOUNT THAT EXCEEDS BUDGET


ACCOUNT    GHI              CATEGORIES EXCEEDING BUDGET
CATEGORY         YTDSPENT        BDGTD AMT     AMOUNTOVER
NO CATEGORIES HAVE AN AMOUNT THAT EXCEEDS BUDGET
```

11

The Sort-Merge Module

DEFINITION

The purpose of the sort-merge module is to order one or more data files, or combine two or more identically ordered files, according to a set of user-specified keys contained within each record.

To accomplish SORT or MERGE operations, the program must use the SELECT clause in the ENVIRONMENT division, the sort file description (SD) entry in the DATA division, and the SORT or MERGE statement in the PROCEDURE division.

A program may apply some special processing to each record before or after a SORT or after a MERGE operation has been completed. The special processes are in input or output procedures that contain RELEASE or RETURN statements, respectively, and that are named in the SORT statement. (MERGE allows only output procedures and the RETURN statement.)

RELEASE, RETURN, and other statements in input and output procedures allow processing not provided by the SORT or MERGE statement alone. They are helpful in creating or selecting certain records to be sorted, and in processing sorted or merged records in memory.

Strategy

For files containing many records, the use of input and output procedures will affect runtime performance of the COBOL program. Thus, if no special operations such as selecting records sent to SORT or selecting records to be written to the output file are desired, it is highly advantageous to specify USING and GIVING instead of input and output procedures in SORT and MERGE statements.

LOADING SORT AND MERGE PROGRAMS

When a runfile is made that includes a SORT or MERGE statement, the sort library (VSRTLI) must be loaded as shown in Chapter 3. An example of loading is given at the end of this chapter.

ENVIRONMENT DIVISION

I-O CONTROL

The I-O-CONTROL paragraph specifies the memory area that is to be shared by different files.

Format

I-O CONTROL.

```
[ ; SAME [ RECORD     ] AREA FOR file-name-1 {, file-name-2} ... ] ...
         [ SORT       ]
         [ SORT-MERGE ]
```

Syntax Rules

1. In the SAME AREA clause, SORT and SORT-MERGE are equivalent.

2. If the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file-names must represent a sort or merge file. Files that do not represent sort or merge files may also be named in the clause.

3. More than one SAME clause may be included in a program, however:

   - A file-name must not appear in more than one SAME RECORD AREA clause.
   - A file-name that represents a sort or merge file must not appear in more than one SAME SORT AREA or SAME SORT-MERGE AREA clause.
   - If a file-name that does not represent a sort or merge file appears in a SAME AREA clause and in one or more SAME SORT AREA or SAME SORT-MERGE AREA clauses, all of the files named in the first clause must be named in the second clause(s).

4. The files referenced in the SAME SORT AREA, SAME SORT-MERGE AREA, or SAME RECORD AREA clause need not all have the same organization or access.

5. SAME AREA is treated as SAME RECORD AREA by this compiler.

General Rules

1. The SAME RECORD AREA or SAME AREA clause specifies that two or more files are to use the same memory area for processing of the current logical record. All of the files may be open at the same time. A logical record in the SAME RECORD AREA is considered as a logical record of each opened output file whose file-name appears in this SAME RECORD AREA clause and of the most recently read input file whose file-name appears in this SAME RECORD AREA clause. This is equivalent to implicit redefinition of the area; records are aligned on the leftmost character position.

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

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

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

   - Files other than sort or merge files do not share the same storage area with each other. If the user wishes these files to share the same storage area with each other, the program must also include a SAME AREA or SAME RECORD AREA clause naming these files.

   - During the execution of a SORT or MERGE statement that refers to a sort or merge file named in this clause, any non-sort-merge files named in the same clause must be closed.

DATA DIVISION

FILE SECTION

An SD file description gives information about the sizes and the names of the data records associated with the file to be sorted. There are no label procedures that the user can control, and the rules for blocking and internal storage are peculiar to the SORT statement.

SORT FILE DESCRIPTION

The sort-merge file description furnishes information concerning the physical structure, identification, and record names of the file to be sorted or merged.

Format

```
SD file-name [ COMPRESSED   ]
             [ UNCOMPRESSED ]

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

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

{record-description-entry}
```

Syntax Rules

1. The level indicator SD identifies the beginning of the sort-merge file description. An FD level indicator must precede the file-name of each file providing input or output to the sort or merge operation.

2. The clauses that follow the file-name are optional, and their order of appearance is immaterial.

3. One or more record-description-entries must follow the file description; however, no I-O statements may be executed for this file.

4. The file must be specified in a SELECT clause.

PROCEDURE DIVISION

MERGE

The MERGE statement combines two or more identically sequenced files on a set of specified keys, and during the process makes records available, in merge order, to an output procedure or to an output file.

Format

```
MERGE file-name-1

    ON { ASCENDING  } KEY data-name-1 [, data-name-2 ···
       { DESCENDING }

  [ ON { ASCENDING  } KEY data-name-3 [, data-name-4] ··· ] ···
       { DESCENDING }

  [COLLATING SEQUENCE IS alphabet-name]

  USING file-name-2, file-name-3 [, file-name-4] ···

  { OUTPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ] }
                                       { THRU    }
  { GIVING file-name-5                                               }
```

Syntax Rules

1. Each file-name-1 must be described in a sort-merge file-description-entry in the DATA division.

2. Each section-name-1 represents the name of an output procedure.

3. Each file-name-2, file-name-3, file-name-4, and file-name-5 must be described in a file-description-entry, not in a sort-merge file-description-entry, in the DATA division.

4. The actual size of the logical record(s) described for file-name-2, file-name-3, file-name-4, and file-name-5 must be equal to the actual size of the logical record(s) described for file-name-1. It is the programmer's responsibility to describe the corresponding records in such a manner as to cause an equal number of character positions to be allocated.

5. The words THRU and THROUGH are equivalent.

6. Data-names 1, 2, 3, and 4 are KEY data-names and are subject to the following rules:

   - The data items identified by KEY data-names must be described in records associated with file-name-1.
   - KEY data-names may be qualified.
   - The data items identified by KEY data-names must not be variable-length items.
   - If file-name-1 has more than one record description, then the data items identified by KEY data-names need be described in only one of the record descriptions.
   - None of the KEY data-names can be described by an entry that either contains an OCCURS clause or is subordinate to an entry that contains an OCCURS clause.

7. File-names must not be repeated within the MERGE statement.

8. MERGE statements may appear anywhere except in the declaratives portion of the PROCEDURE division or in an input or output procedure associated with a SORT or MERGE statement.

9. The number of files allowed in the USING or GIVING list is given in Appendix J. All files in this list must have the same type, either COMPRESSED or UNCOMPRESSED.

10. The alphabet-name may be NATIVE, STANDARD-1, or EBCDIC. NATIVE and STANDARD-1 both mean the ASCII standard, which is also the default. More discussion of these collating sequences is given with SORT below.

General Rules

1. The MERGE statement will merge all records contained in file-name-2, file-name-3, and file-name-4. These files are automatically opened and closed by the merge operation with all implicit functions performed, such as the execution of any associated USE procedures. The terminating function for all files is performed as if a CLOSE statement had been executed for each file.

   Files referenced in a MERGE statement must be closed prior to execution of the merge and may not be opened by the user until the merge operation is complete. However, a file named in an output procedure must have been opened by an explicit OPEN statement and subsequently written, probably in the output procedure, and then explicitly CLOSED.

2. The data-names following the word KEY are listed from left to right in the MERGE statement in order of decreasing significance without regard to how they are divided into KEY phrases. In the format, data-name-1 is the major key, data-name-2 is the next most significant key, and so on.

    - When the ASCENDING phrase is specified, the merged sequence will be from the lowest value of the KEY data-names to the highest value, according to the rules for comparison of operands in a relation condition in Chapter 4.

    - When the DESCENDING phrase is specified, the merged sequence will be from the highest value of the KEY data-names to the lowest value, according to the rules for comparison of operands in a relation condition in Chapter 4.

3. The collating sequence that applies to the comparison of the nonnumeric key data items specified is determined in the following order of precedence:

    - First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in that MERGE statement.

    - Second, the collating sequence established as the program collating sequence.

4. The output procedure must consist of one or more sections that appear contiguously in a source program and do not form a part of any other procedure. In order to make merged records available for processing, the output procedure must include the execution of at least one RETURN statement. Control must not be passed to the output procedure except when a related SORT or MERGE statement is being executed. The output procedure may consist of any procedures needed to select, modify, or copy the records that are being returned one at a time in merged order, from file-name-1. The restrictions on the statements within the output procedure are as follows:

    - The output procedure must not contain any transfers of control to points outside the output procedure; ALTER, GO TO, and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside it. COBOL statements are allowed that will cause an implied transfer of control to declaratives.

    - The output procedures must not contain any SORT or MERGE statements.

- The remainder of the PROCEDURE division must not contain any transfers of control to points within the output procedures; ALTER, GO TO, and PERFORM statements in the remainder of the PROCEDURE division are not permitted to refer to procedure-names within the output procedures.

5. If an output procedure is specified, control passes to it during execution of the MERGE statement. The compiler inserts a return mechanism at the end of the last section in the output procedure. When control passes the last statement in the output procedure, the return mechanism provides for termination of the merge, and then passes control to the next executable statement after the MERGE statement. Before entering the output procedure, the merge procedure reaches a point at which it can select the next record in merged order when requested. The RETURN statements in the output procedure are the requests for the next record.

6. If the GIVING phrase is specified, all the merged records in file-name-1 are automatically written on file-name-5 as the implied output procedure for this MERGE statement.

7. In the case of identical key fields between records from two or more input files the records are written on file-name-5 or returned to the output procedure in the order that the associated input files are specified in the MERGE statement.

8. The results of the merge operation are predictable only when the records in the files referenced by file-name-2, file-name-3, ..., are ordered as described in the ASCENDING or DESCENDING KEY clause associated with the MERGE statement.

Example

The following program merges two files, MRGFILE1 and MRGFILE2, using an output procedure that sends them to the print file YEARLY. An example of loading and execution and the sample files follow the program.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MERGSAMP.
AUTHOR. W. T. CARBONNEAU.
INSTALLATION. PRIME.
DATE-WRITTEN. 14 APRIL 82.
DATE-COMPILED.
SECURITY. N/A.
REMARKS. TESTING THE MERGE VERB. FILES MUST BE SORTED FIRST!
*****************************************************************
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. PRIME.
OBJECT-COMPUTER. PRIME.
INPUT-OUTPUT SECTION.
```

```
 FILE-CONTROL.
     SELECT FIRST-HALF ASSIGN TO PFMS.
     SELECT SECOND-HALF ASSIGN TO PFMS.
     SELECT YEARLY ASSIGN TO PRINTER.
     SELECT MERGE-FILE ASSIGN TO PFMS.
*
 DATA DIVISION.
 FILE SECTION.
 FD  FIRST-HALF
     LABEL RECORDS ARE STANDARD
     VALUE OF FILE-ID IS 'MRGFIL1'
     DATA RECORD IS SALES-HISTORY-1.
 01  SALES-HISTORY-1.
     05 DEPT-NO          PIC 999.
     05 PROD-NO          PIC 9(5).
 FD  SECOND-HALF
     LABEL RECORDS ARE STANDARD
     VALUE OF FILE-ID IS 'MRGFIL2'
     DATA RECORD IS SALES-HISTORY-2.
 01  SALES-HISTORY-2.
     05 DEPT-NO          PIC 999.
     05 PROD-NO          PIC 9(5).
 FD  YEARLY
     LABEL RECORDS ARE STANDARD
     DATA RECORD IS CUMULATIVE-SALES.
 01  CUMULATIVE-SALES    PIC 9(8).
 SD  MERGE-FILE
     DATA RECORD IS MERGE-RECORD.
 01  MERGE-RECORD.
     05 DEPARTMENT       PIC 999.
     05 PRODUCT          PIC 9(5).
*
 WORKING-STORAGE SECTION.
 01 END-OF-DATA          PIC XXX     VALUE 'NO '.
 01  PROOF-LIST.
     05 DEPT-NO-REPORT   PIC 999.
     05 FILLER-1         PIC XXX.
     05 PROD-NO-REPORT   PIC 9(5).
*
 PROCEDURE DIVISION.
     MERGE MERGE-FILE ON ASCENDING KEY DEPARTMENT
         USING FIRST-HALF, SECOND-HALF
             OUTPUT PROCEDURE IS OUTPUT-PROCEDURE.
     STOP RUN.
*
 OUTPUT-PROCEDURE SECTION.
 CREATE-PROOF-LIST.
     OPEN OUTPUT YEARLY.
     PERFORM RETURN-DATA.
     PERFORM WRITE-DATA UNTIL END-OF-DATA = 'YES'.
     CLOSE YEARLY.
     EXIT.
```

```
       RETURN-DATA SECTION.
           RETURN MERGE-FILE INTO PROOF-LIST AT END
               MOVE 'YES' TO END-OF-DATA.
      *
       WRITE-DATA SECTION.
           MOVE SPACES TO FILLER-1.
           WRITE CUMULATIVE-SALES FROM PROOF-LIST AFTER ADVANCING 1.
           PERFORM RETURN-DATA.
```

First Input File

```
00123576
00376231
00592862
```

Second Input File

```
00263550
00443651
00640166
```

Compiling, Loading, and Executing

The program may be compiled, loaded, and run with the following dialog. The error message 247 is displayed only to encourage the programmer to check that the output procedure was coded correctly.

```
OK, CBL MERGSAMP

[CBL rev xx]

ERROR 247  SEVERITY 1  LINE 73  COLUMN 19  [OBSERVATION, SEMANTICS]
The section that immediately contains the RETURN statement was not
named as an output procedure associated with a SORT or MERGE
statement. Check that the perform range of the applicable procedure
contains this section.

[1 OBSERVATION in program: <OPERSY>ANNE.K>NEWCBL>MERGSAMP.CBL]

OK, SEG -LOAD
[SEG rev x.x]
$ LO MERGSAMP
$ LI CBLLIB
$ LI VSRTLI
$ LI
LOAD COMPLETE
$ Q
OK,
```

```
.
.
OK, SEG MERGSAMP
OK,
```

Output File

```
01   76
02   50
03   31
04   51
05   62
06   66
```

RELEASE

The RELEASE statement transfers records to the initial phase of a SORT operation, allowing processing of the record content.

Format

RELEASE record-name [FROM data-name]

Syntax Rules

1. A RELEASE statement may be specified only within an input procedure associated with a SORT statement for a file whose SD entry contains record-name. Input procedures are described with the SORT statement below.

2. The record-name must be the name of a logical record in the associated SD entry. The record-name may be qualified.

3. The data-name must be a data-name containing a record read from an input file (FD entry). It may be in WORKING-STORAGE.

General Rules

1. The execution of a RELEASE statement causes the record-name to be released to the initial phase of a SORT operation.

   A RELEASE statement must be executed for each record to be sent to a sort or merge operation.

2. If the FROM phrase is used, the contents of the data-name are moved to the record-name, then the contents of the record-name are released to the sort file. Moving takes place according to the rules for the MOVE statement without the CORRESPONDING phrase.

3. After the execution of the RELEASE statement, the logical record is still available as a record of other files referenced in the SAME AREA clause, as well as to the file associated with record-name. When control passes from the input procedure, the file consists of all those records placed in it by the execution of RELEASE statements.

4. If a RELEASE statement releases a record associated with an input file, the input file must have been opened and read.

Example

This input procedure is incorporated in the sample program at the end of this chapter.

```
 FD  IN-FILE, COMPRESSED,
     LABEL RECORDS ARE STANDARD,
     VALUE OF FILE-ID IS 'BUDGET'.
     01  ENTRY                    PIC X(80).
*
 FD  OUT-SORT,
     LABEL RECORDS ARE STANDARD,
     RECORD CONTAINS 80 CHARACTERS.
 01  SORTOUT                      PIC X(80).
*
 SD  SORT-WK,
     RECORD CONTAINS 80 CHARACTERS.
 01  SORT-REC.
     05 CODE-SD                   PIC X.
     05 ACCT-SD                   PIC X(3).
     05 CAT-SD                    PIC XX.
     05 DATE-SD                   PIC 9(6).
     05 FILLER                    PIC X(68).
       .
       .
       .
 030-INPUT-PROC SECTION.
     OPEN INPUT IN-FILE.
     READ IN-FILE
         AT END DISPLAY 'EMPTY FILE' MOVE 'Y' TO
                NO-MORE-RECORDS.
     PERFORM 035-ERROR-CHECK UNTIL NO-MORE-RECORDS = 'Y'.
     CLOSE IN-FILE
     EXIT.
*
 035-ERROR-CHECK SECTION.
     IF ACCT-CD NUMERIC,
         DISPLAY '**ERROR: ***', DISPLAY ENTRY,
     ELSE RELEASE SORT-REC FROM ENTRY.
     READ IN-FILE INTO ENTRY,
         AT END DISPLAY 'END OF FILE', MOVE 'Y' TO
             NO-MORE-RECORDS.
     EXIT.
```

RETURN

The RETURN statement obtains sorted records from the final phase of a sort operation, or merged records during a merge.

Format

RETURN file-name RECORD [INTO data-name]

; AT END imperative-statement

Syntax Rules

1. The file-name must be described by an SD entry in the DATA division.

2. The data-name must be able to contain a record to be written to an output file.

3. A RETURN statement may be specified only within an output procedure associated with a SORT or MERGE statement for file-name. Output procedures are described with the SORT and MERGE statements.

4. The INTO phrase must not be used if the input file contains logical records of various sizes.

5. The areas associated with data-name and file-name may be the same storage area.

General Rules

1. If more than one record description is associated with file-name, these records automatically share the same storage area; that is, the area is implicitly redefined. After the execution of the RETURN statement, any data items which lie beyond the range of the current record are undefined.

2. When the RETURN statement is executed, the next record from file-name (in the order of the key) is made available for processing in the record areas associated with the sort or merge file.

   A RETURN statement must be executed for each record to be retrieved from the sort or merge operation.

3. If the INTO phrase is specified, the current record is moved from the input (file) area to the area specified by data-name according to the rules for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if there is an AT END condition. Any subscripting or indexing associated with data-name is evaluated after the record has been returned and immediately before it is moved to the data-name.

4. When the INTO phrase is used, the data is available in both the input record area and the data area associated with data-name.

5. If no next logical record exists at the execution of a RETURN statement, the AT END condition occurs. The contents of the record areas associated with the file are undefined when that condition occurs. After the execution of the imperative-statement in the AT END phrase, no RETURN statement may be executed as part of the current output procedure.

SORT

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

Format

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

[ ON { ASCENDING  } KEY data-name-3 [, data-name-4] ... ] ...
     { DESCENDING }

[COLLATING SEQUENCE IS alphabet-name]

{ INPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ] }
{                                     { THRU    }                 }
{ USING file-name-2 [, file-name-3] ...                           }

{ OUTPUT PROCEDURE IS section-name-3 [ { THROUGH } section-name-4 ] }
{                                      { THRU    }                  }
{ GIVING file-name-4                                                }
```

Syntax Rules

1. SORT statements may appear anywhere in the PROCEDURE division except in the DECLARATIVES portion of the PROCEDURE division or in an input or output procedure associated with a SORT or MERGE statement.

2. File-name-1 must be described in an SD entry in the DATA division. File-names 2, 3, and 4 must be described in FD entries.

3. If the USING phrase is specified and file-name-1 contains variable-length records, the size of the records contained in file-name-2 must not be less than the smallest record nor larger than the largest record described for file-name-1. If file-name-1 contains fixed-length records, the size of the records contained in file-name-2 must not be larger than the largest record described for file-name-1.

4. Data-names 1, 2, and so on (KEY data-names) are subject to the following rules:

   - The data items identified by KEY data-names must be described in records associated with file-name-1.
   - KEY data-names may be qualified.
   - KEY data-names may not describe variable-length data items, nor may they name group items that contain variable-occurrence data items.
   - If file-name-1 has more than one record description, then the data items identified by KEY data-names need be described in only one of the record descriptions. In other words, the same character positions referenced by a KEY data-name in one record-description-entry are taken as the KEY in all records of file-name-1.
   - The data items identified by KEY data-names may not contain an OCCURS clause or be subordinate to an item that contains an OCCURS clause.

5. The section-name-1 specifies the first section in an input procedure. The section-name-2, if specified, indentifies the last section of an input procedure.

   The section-names 3 and 4 name an output procedure.

6. The words THRU and THROUGH are equivalent.

7. In the DATA division, file-name-2, file-name-3, and file-name-4 must be described in an FD entry, not in an SD entry.

8. The actual size of the logical records described for file-names 2, 3, and 4 must be equal to the actual size of the records for file-name-1. Equal numbers of character positions must be allocated for corresponding records.

9. If the GIVING phrase is specified and file-name-3 contains variable-length records, the size of the records contained in file-name-1 must not be less than the smallest record nor larger than the largest record described for file-name-3. If file-name-3 contains fixed-length records, the size of the records contained in file-name-1 must not be larger than the largest record described for file-name-3.

10. The number of files that may be sorted from the USING list is given in Appendix J. All files contained in the USING list must have the same type, COMPRESSED or UNCOMPRESSED.

11. SORT and MERGE are not supported for tape.

General Rules

1. Files referenced in a SORT statement must be closed prior to execution of the sort operation and may not be opened by the user, except through an input or output procedure, until after the sort is complete.

2. If file-name-1 contains only fixed-length records, any record in file-name-2 released to file-name-1 is left justified, and any unused character positions at the right end of the record are filled with blanks.

3. The data-names following the word KEY are listed in order of decreasing significance no matter how they are divided into KEY phrases. For example, data-name-1 is the major key, data-name-2 is the next most significant key.

    - When the ASCENDING phrase is specified, the sorted sequence will be from the lowest key value to the highest key value.

    - When the DESCENDING phrase is specified, the sorted sequence will be from the highest key value to the lowest key value.

    - The key values are compared according to the rules for comparison of operands in a relation condition. (See CONDITIONAL EXPRESSIONS in Chapter 4.)

4. If the contents of all KEY data items associated with two or more data records are equal, then the order of return for the records is undefined.

5. The files specified in USING and GIVING must have sequential organization.

6. The collating sequence used in sorting nonnumeric items is determined in the following order of precedence:

   - The sequence established by the COLLATING clause, if any, in the SORT statement
   - The program collating sequence

   As an example of the difference that the COLLATING sequence can make, consider the following file.

```
OK,   SLIST COLLATING.DATA
BABC0101327803002OO
AABC0001234567OOOOO
2OOC0200432987654OO
```

   If this file is sorted with COLLATING SEQUENCE IS NATIVE, COLLATING SEQUENCE IS STANDARD-1, or no COLLATING SEQUENCE clause, the output file is the following.

```
OK,   SLIST F2.NATIVE
2OOC0200432987654OO
AABC0001234567OOOOO
BABC0101327803002OO
```

   With COLLATING SEQUENCE IS EBCDIC, the output file looks like this.

```
OK,   SLIST F2.EBCDIC
AABC0001234567OOOOO
BABC0101327803002OO
2OOC0200432987654OO
```

Rules for Input Procedures and USING

1. The input procedure must consist of one or more sections that are written consecutively and do not form a part of any output procedure. In order to transfer records to file-name-1, the input procedure must include at least one RELEASE statement. Control must not be passed to the input procedure except when a related SORT statement is being executed. An example is given at the end of this chapter.

The input procedure can include any procedures needed to select, create, or modify records, including a READ for the input file, which must first be opened. There are three restrictions on the statements within the input procedure:

- The input procedure must not contain any SORT or MERGE statements.
- The input procedure must not contain any explicit transfers of control to points outside the input procedure; GO TO and PERFORM statements in the input procedure are not permitted to refer to procedure-names outside it. COBOL statements are allowed that will cause an implied transfer of control to a declarative section.
- The remainder of the PROCEDURE division must not contain any transfers of control to points inside the input procedure; GO TO and PERFORM statements in the remainder of the PROCEDURE division must not refer to procedure-names within the input procedure.

2. If an input procedure is specified, control is passed to the input procedure before file-name-1 is sorted by the SORT statement. When control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted.

3. If the USING phrase is specified, all the records in the USING file list (file-name-2 and so on) are automatically transferred to file-name-1. At the time of execution of the SORT statement, files in the USING list must not be open. For files in the USING list, the execution of the SORT statement causes the following actions to be taken:

   - The processing of the file is initiated as if an OPEN statement with the INPUT phrase had been executed.
   - The file references are passed to the sort routine, which puts all of the records in a single file. Each record is obtained as if a READ statement with the NEXT and the AT END phrase had been executed.
   - The processing of the file is terminated as if a CLOSE statement had been executed.

4. SORT USING is not supported for tape.

Rules for Output Procedures and GIVING

1. The output procedure must consist of one or more sections that are written consecutively and do not form a part of any input procedure. In order to make sorted records available for processing, the output procedure must include at least one RETURN statement. Control must not be passed to the output procedure except when a related SORT statement is being executed. An example is given with the discussion of MERGE above.

   The output procedure may consist of any procedures needed to select, modify, or copy the records that are being returned, one at a time in sorted order, from the sort file. It may include a WRITE statement for the output file, which must first be opened. There are three restrictions on the procedural statements within the output procedure:

   - The output procedure must not contain any SORT or MERGE statements.
   - The output procedure must not contain any explicit transfers of control to points outside it; GO TO and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside it. COBOL statements are allowed that will cause an implied transfer of control to declarative sections.
   - The remainder of the PROCEDURE division must not contain any transfers of control to points within the output procedure; GO TO and PERFORM statements in the remainder of the PROCEDURE division must not refer to procedure-names within the output procedure.

2. If an output procedure is specified, control passes to it after file-name-1 has been sorted by the SORT statement. When control passes the last statement in the output procedure, control returns to the next executable statement after the SORT statement. Before entering the output procedure, the sort procedure reaches a point at which it can select the next record in sorted order, when requested. The RETURN statements in the output procedure are the requests for the next record.

3. If the GIVING phrase is specified, all the sorted records are automatically written on file-name-4 as the implied output procedure for the SORT statement. At the time of the execution of the SORT statement, file-name-4 must not be open. For file-name-4, the execution of the SORT statement causes the following actions to be taken:

   - The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase had been executed.

   - The sorted logical records are returned and written onto the file. The records are written as if a WRITE statement without any optional phrases had been executed.

   - The processing of the file is terminated. The termination is performed as if a CLOSE statement had been executed.

4. If file-name-4 contains only fixed-length records, any record in file-name-1 containing fewer character positions is padded with blanks at the right end of the record when the record is returned to file-name-4.

EXAMPLE

A source file for sample program SAMPLE.SORT.CBL is presented below. This example uses a SORT statement with an input procedure. The input procedure edits records for errors before releasing them to the sort file SORT-WK.

```
           EXIT.
       IDENTIFICATION DIVISION.
       PROGRAM-ID.                     SRTBUDGT.
       AUTHOR.                         PEGGY PECK.
       INSTALLATION.                   PRIME.
       DATE-COMPILED.
      ******************************************************************
*
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. PRIME.
       OBJECT-COMPUTER. PRIME.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT IN-FILE ASSIGN TO PFMS,
               FILE STATUS IS FILE-STAT.
           SELECT OUT-SORT ASSIGN TO PFMS.
           SELECT SORT-WK ASSIGN TO PFMS.
      ******************************************************************
*
       DATA DIVISION.
       FILE SECTION.
      *
       FD  IN-FILE,COMPRESSED,
           LABEL RECORDS ARE STANDARD,
           VALUE OF FILE-ID IS 'SEFILE'.
       01  ENTRY.
           05 CODE-IN                  PIC X.
           05 ACCT-IN                  PIC X(3).
           05 FILLER                   PIC X(76).
      *
       FD  OUT-SORT,
           LABEL RECORDS ARE STANDARD,
           RECORD CONTAINS 80 CHARACTERS.
       01  SORTOUT                     PIC X(80).
      *
       SD  SORT-WK,
           RECORD CONTAINS 80 CHARACTERS.
       01  SORT-REC.
           05 CODE-SD                  PIC X.
           05 ACCT-SD                  PIC X(3).
           05 CAT-SD                   PIC XX.
           05 DATE-SD                  PIC 9(6).
           05 FILLER                   PIC X(68).
      *
       WORKING-STORAGE SECTION.
       77  FILE-STAT                   PIC XX.
```

```
 77  NO-MORE-RECORDS              PIC X  VALUE 'N'.
*****************************************************************
 PROCEDURE DIVISION.
*
 DECLARATIVES.
 INPUT-ERROR SECTION.  USE AFTER ERROR PROCEDURE ON IN-FILE.
   FIRST-PARAGRAPH.
     DISPLAY 'ERROR ON INPUT FILE - END OF RUN',
         DISPLAY 'FILE STATUS IS ', FILE-STAT,
         STOP RUN.
 END DECLARATIVES.
*
 MAINLINE SECTION.
 000-MAINLINE.
     PERFORM 020-SORT-TRANSACTIONS.
     STOP RUN.
*
 020-SORT-TRANSACTIONS.
     SORT SORT-WK ASCENDING KEY CODE-SD,
         DESCENDING KEY ACCT-SD,
         ASCENDING KEY CAT-SD,
         INPUT PROCEDURE IS 030-INPUT-PROC,
         GIVING OUT-SORT.
*
 030-INPUT-PROC SECTION.
      OPEN INPUT IN-FILE.
      READ IN-FILE INTO ENTRY,
         AT END DISPLAY 'EMPTY FILE'
         MOVE 'Y' TO NO-MORE-RECORDS.
      PERFORM 035-ERROR-CHECK UNTIL NO-MORE-RECORDS
          = 'Y'.
      CLOSE IN-FILE.
*
   035-ERROR-CHECK SECTION.
      IF ACCT-IN NOT NUMERIC,
          DISPLAY '**ERROR: ***',
          DISPLAY ENTRY,
      ELSE RELEASE SORT-REC FROM ENTRY.
     READ IN-FILE INTO ENTRY,
          AT END DISPLAY 'END OF FILE' MOVE 'Y'
          TO NO-MORE-RECORDS.
     EXIT.
```

Below are a sample input file (SEFILE) including one erroneous entry, a sample dialog for compiling, loading, and running the program, and the resulting output file (OUT-SORT).

Input file

```
1003JOSEPH BLOUGH        99 12345678
3007JOSEPHINE BLOUGH     66 12345678
2002JOSE BLOUGH          00 12345678
4AZ5JOSIP BLOUGH         33 12345678
```

Compiling, Loading, and Execution

```
OK, CBL SORT2 -LIST

[CBL rev 9]

ERROR 247  SEVERITY 1  LINE 82  COLUMN 26  [OBSERVATION, SEMANTICS]
The section that immediately contains the RELEASE statement was not
named as an input procedure associated with a SORT or MERGE
statement.  Check that the perform range of the applicable
procedure contains this section.

ERROR 262  SEVERITY 1
File "SORT-WK" was accessed by an INPUT verb but never opened for I
NPUT.

[2 OBSERVATIONS in program: <OPERSY>ANNE.K>NEWCBL>SORT2.CBL]

OK, SEG -LOAD
[SEG rev x.x]
$ LO SORT2
$ LI CBLLIB
$ LI VSRTLI
$ LI
LOAD COMPLETE
$ EXEC


**ERROR: ***
4AZ5JOSIP BLOUGH         33 12345678
END OF FILE
OK,
```

Sorted Output File (OUT-SORT)

```
1003JOSEPH BLOUGH        99 12345678
2002JOSE BLOUGH          00 12345678
3007JOSEPHINE BLOUGH     66 12345678
```

12
Indexed Sequential Files

FUNCTION OF THE INDEXED I-O MODULE

The Indexed I-O module allows random or sequential access of records of an indexed disk file. Each record in an indexed file is uniquely identified by the value of one or more keys within that record.

The Prime COBOL Indexed I-O module is intended to be used with indexed files created with the MIDAS or MIDAS utility. Each COBOL record key corresponds to a MIDAS index. How to prepare files for COBOL access with MIDAS is discussed in Appendix E. The subject is presented in more detail in the section on COBOL in the MIDAS User's Guide.

> Caution
>
> Do not use COBOL with an outdated revision of MIDAS or MIDASPLUS.

LOADING AND EXECUTING PROGRAMS WITH RELATIVE FILES

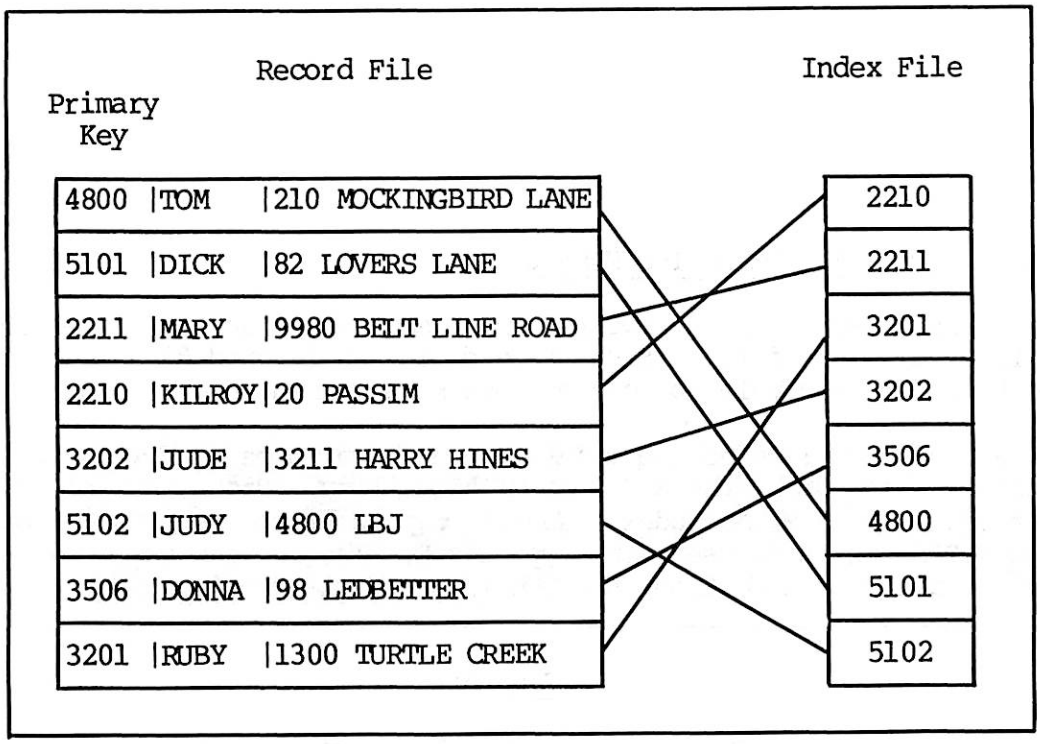
Use the SEG command steps in Chapter 3 to load a runfile. An example of loading is given at the end of this chapter.

INDEXED FILE CONCEPTS

Organization

An indexed file is a disk file in which data records may be accessed by the value of a key. A record description must include one or more data items used as keys, each of which is associated with a MIDAS index.

The indexes are created in any order on a disk, but also one or more files of indexes are constructed. All access to these files is thus done according to the value of the key field related to one of the file's indexes. More discussion of index files and data files is presented in the MIDAS User's Guide. A representation of a MIDAS indexed file together with a file of indexes is presented in Figure 12-1.



MIDAS Indexed Record File and File of Indexed
Figure 12-1

Primary and Secondary Keys

For inserting, updating, and deleting records in a file, each record is identified solely by the value of a record key. The data item named in the RECORD KEY clause of a file-control-entry for a file is the primary record key for that file. Secondary keys may be designated with the ALTERNATE RECORD KEY clause.

A secondary or alternate record key corresponds to a MIDAS secondary index.

A file used as an indexed file in a program must have a template created as an indexed file with MIDASPLUS. That is, the file's structure is created with MIDASPLUS and then the file is filled with records, either with MIDASPLUS or with a COBOL program. All fields, keys, secondary keys, and keys allowing duplicates must have been declared as such when the file template was created; they cannot be changed merely by their description in the COBOL program.

Access Modes

Three access modes are possible in COBOL for indexed files. They are specified in the SELECT clause:

- In sequential access mode, records are accessed in ascending order of record key values. In the case of duplicate key values (for secondary keys only), the records are retrieved in the order in which they were written to the file.
- In random access mode, the sequence in which records are accessed is controlled by the program. The desired record is accessed by placing the value of its key in the corresponding field of the record-description-entry.
- In dynamic access mode, the programmer may change at will from sequential access to random access for reading the file. Access mode requirements for each I-O statement are discussed in the section COMMON OPERATIONS ON INDEXED FILES later in this chapter.

Current Record Pointer

The current record pointer is a conceptual entity used to indicate the next record to be accessed within a given file. The setting of the current record pointer is affected only by the OPEN, START, DELETE, and READ statements.

File Status

A file status check should be coded in the program to determine the success or failure of an I-O operation. The result can be used to control the next action. If the FILE STATUS clause is specified in a file-control-entry, a value is automatically placed into the specified data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE,

DELETE, or START statement to indicate the status of that input-output operation. FILE STATUS is discussed below with the ENVIRONMENT division. A list of file status codes is given in Table A-5 of Appendix A.

The INVALID KEY Condition

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

When the INVALID KEY condition is recognized, these actions occur 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, but a USE procedure is specified, either explicitly or implicitly, for this file, that procedure is executed.

When the INVALID KEY condition occurs, execution of the input-output statement that caused the condition is unsuccessful and the file is not affected.

The AT END Condition

The AT END condition can occur as a result of a READ statement. For details of the causes of the condition, see the READ statement.

COMMON OPERATIONS ON INDEXED FILES

Files must be opened with the OPEN statement before any other I-O statements are executed, and must then be closed with CLOSE before the program ends. A file must also be closed before being reopened in another mode of operation.

The concept of record key is essential for most of the following operations on indexed files, since MIDAS recognizes only the keys or indexes, and knows nothing about the order or location of the data records themselves.

Create a File

Use the MIDAS CREATK command to create an index template. Then build a file on that template either with the MIDAS KBUILD command and an existing data file, or with a COBOL program. Indexed files cannot be created by a COBOL program until a MIDAS template exists.

The COBOL program must describe the new file as indexed, and open it in whichever access mode is desired. Before each record is written, a unique value must be placed in the record key field.

In all access modes, records may be inserted in any order. In all modes, the proper value must be in the primary key field.

Position the File to a Certain Record

For sequential access or for dynamic access on sequential files, START must be used to position the file if the first record is not desired. If groups of records within the file are to be read sequentially, more than one START may be used.

In random access mode, all reads use the primary key unless the KEY IS clause is included. START is not used in random access mode.

Read a Certain Record

If the file is opened in sequential access, to read from the start of the file in sequential order of the primary key no special operations other than READ are necessary. If the first record is not desired or if a secondary key order is desired, the first READ should be preceded by START to specify the key of the first record sought. (START is required to establish a secondary key.) All subsequent READs will access each next record in key order until another START is used or an error occurs.

If random access is used, all READs use the primary key unless the KEY IS clause is included.

If access will be both sequential and random, dynamic access mode should be specified. To change from random to sequential reading, use a series of Format-1 READs (READ NEXT or implied READ NEXT).

Establish a Key

The default key is the primary key for any I-O verb. To establish a secondary key, use START KEY IS ... data-name or READ ... KEY is data-name.

Delete a Certain Record

DELETE is used in all access modes. In sequential access mode, the record must first be read. In random or dynamic access, the proper value must be placed in the key field before the DELETE.

Update (Change) a Certain Record

The REWRITE statement is used, and the file must be opened for I-O. Before a rewrite in any access mode, the record must first have been read to assure that it cannot be updated by another program running concurrently.

Create (Add) Records

New records are added with WRITE. In all access modes, records may be inserted in any order (a Prime extension). The proper value must be in the primary key field.

Handle Errors in All of These Statements

The INVALID KEY clause and the USE statement in the declarative section provide error handling. One of these elements is required for an I-O statement. AT END is used only for READ in SEQUENTIAL mode.

ENVIRONMENT DIVISION

This section stresses information that is unique to indexed files. Information for sequential files and for relative files is given in Chapters 6 and 13, respectively.

INPUT-OUTPUT SECTION -- FILE CONTROL

Function

The FILE-CONTROL paragraph names each file and specifies other file-related information.

Format

SELECT file-name ASSIGN TO PFMS

; ORGANIZATION IS INDEXED

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

; RECORD KEY IS data-name-1

[; ALTERNATE RECORD KEY IS data-name-2 [WITH DUPLICATES]] ...

[; FILE STATUS IS data-name-3].

General Rules

1. The SELECT clause specifies the name of the indexed sequential file. The SELECT clause must be specified first in the file-control-entry. The remaining clauses may appear in any order. All indexed files are Prime File Management System (PFMS) disk files.

2. The ORGANIZATION IS INDEXED clause specifies that the file named in the SELECT clause contains data organized by indexes, and that it is to be processed by MIDAS. The file organization is established at the time the file is created and cannot be changed by this clause.

3. The ACCESS MODE clause specifies how an indexed file is to be written or read. If it is omitted, sequential access is implied. There are three access modes:

- When SEQUENTIAL is specified, records will be retrieved in the order of ascending values for a given key field.

  Prime extension: records may be written in any order according to the value of the record key.

- When RANDOM is specified, the records are to be written or retrieved randomly only, based on the value placed in the RECORD KEY field prior to a READ or WRITE. The complete RECORD KEY value must be placed in data-name-1 prior to every access operation; otherwise the record will not be found. Random mode precludes a sequential READ NEXT.

- When DYNAMIC access method is specified, a program can read randomly or sequentially. Other operations follow the rules for random access, except that START may be used in dynamic access.

4. The RECORD KEY clause specifies the data item within each record which is used for the primary index.

Note

The primary key (data-name-1) must be within the record-description-entry, and must be the first field in the entry. The value in the primary key must be unique for each record.

- Data-name-1 must be the first entry in the record description associated with the FD entry for the file. It must not be of variable size.

  Prime extension: data-name-1 may be either alphanumeric or numeric.

  Multiple record-description-entries must have the same data description in the same relative position for the record key.

- Data-name-1 must not be specified with an OCCURS clause, or be contained within a group subordinate to an OCCURS clause. This means it may not be subscripted or indexed, but it may be qualified.

- Prime restriction: data-name-1 must not be specified with a P character or a separator sign (/) in its PICTURE clause. It cannot exceed 32 characters.
- Data-name-1 must have the same description and relative size as when the file template was created with CREATK.
- The value contained within data-name-1 must be unique for each record in a file.

5. The ALTERNATE RECORD KEY clause specifies a data item within each record that is used as a secondary index. There may be up to 17 alternate record keys. The number of alternate record keys cannot be greater than the number of secondary indexes specified when the file was created with CREATK.

   Alternate record keys must be part of the record-description-entry, but they cannot be embedded within nor overlap the primary record key. They follow the rules for RECORD KEY above, except that an alternate key may not be the first field in the record-description-entry and may have duplicates.

   Specification of WITH DUPLICATES documents that secondary keys containing the same value are placed in the file. WITH DUPLICATES should only be specified if duplicates are allowed for the corresponding secondary index in the MIDAS or MIDAS template. (The MIDAS file may be changed with the MODIFY option of CREATK, as explained in Chapter 12 of the MIDAS User's Guide.) If DUPLICATES is not specified, the secondary key value in each record should be unique.

   Secondary keys must be defined in the same order as the corresponding MIDAS indexes.

   Alternate record keys may be nonnumeric (a Prime extension).

6. In the FILE STATUS clause, data-name-3 must be a two-character field described in the DATA division. The file control system moves a value into data-name-3 following the execution of every statement that explicitly or implicitly references the file. This value indicates the execution status of the statement. Following a successful I-O operation, data-name-3 contains '00'. The complete status codes are described in Table A-5 of Appendix A.

   The file status item (data-name-3) may not be part of the record description for its file. It must be in the WORKING STORAGE or LINKAGE section. It may be qualified but must not be subscripted or indexed.

   Prime extension: data-name-3 (FILE STATUS) may be either alphanumeric or numeric (PIC XX or PIC 99).

I-O-CONTROL

Function

The I-O-CONTROL paragraph specifies the points at which rerun is to be established, and the memory area to be shared by different files.

Format

```
I-O CONTROL.

  [ ; SAME [ RECORD ] AREA FOR file-name-1 {,file-name-2} ... ] ...

  [ ; RERUN [ ON file-name-3]

        EVERY { integer-4 RECORDS OF file-name-4 }
              { integer-5 CLOCK-UNITS           }  ...
              { condition-name                  }
  ]
```

General Rules

1. The SAME RECORD AREA or SAME AREA clause specifies that two or more files are to use the same memory area for processing of the current logical record. This saves memory space and eliminates MOVEs from one area to another. An example is included in the sample program at the end of this chapter.

   A logical record in the SAME RECORD AREA is considered both as a logical record of each opened output file in this clause, and as a record of the most recently read input file in this clause. This is equivalent to an implicit redefinition of the area; that is, records are aligned on the leftmost character position.

2. If a file-name in a SAME AREA clause appears in a SAME RECORD AREA clause, all of the file-names in the first clause must appear in the second. However, additional file-names not appearing in the first clause may also appear in the second clause.

3. The files referenced in the SAME AREA or SAME RECORD AREA clause need not all have the same organization or access.

4. On Prime systems, SAME AREA is equivalent to SAME RECORD AREA.

5. The RERUN clause specifies points in the program at which processing can be restarted in case of premature end of execution. The clause is syntax-checked only.

DATA DIVISION

The elements of the DATA division are the same for indexed files as those described in Chapter 7 except for the following two items.

RECORD-DESCRIPTION-ENTRY

The first field in the entry must be the primary key. Each record must have a unique value for the primary key. Rules for primary and secondary keys are given in the ENVIRONMENT section above.

PROCEDURE DIVISION

The COBOL statements listed in this chapter are described only as they apply to indexed file processing. A complete description of all COBOL verbs is provided in Chapter 8, PROCEDURE DIVISION.

CLOSE

Function

The CLOSE statement terminates the processing of files.

Format

CLOSE file-name-1 [, file-name-2] ···

Syntax Rule

The files named in the CLOSE statement need not all have the same organization or access.

General Rule

Once a CLOSE statement has been executed for a file, no other statement can be executed for that file unless an intervening OPEN statement for that file is executed.

DELETE

Function

The DELETE statement logically removes a record from a file.

Format

DELETE file-name RECORD [; INVALID KEY imperative-statement]

Syntax Rule

The INVALID KEY clause must not be specified for a DELETE on a file in sequential access mode. It must be specified for a DELETE on a file that is not in sequential access mode and for which no USE procedure is specified.

General Rules

The DELETE statement logically removes a data record from the indexed file together with all the associated index entries. The file must be open for I-O. Execution of DELETE causes the value of the FILE STATUS data item, if any, to be updated. It does not affect the contents of the record area associated with file-name.

Rules for Sequential Access

1. In sequential access, the record to be deleted must have been successfully read before a delete can be executed.

2. The primary record key cannot be changed between the READ and DELETE statements. Otherwise the INVALID KEY condition will occur, and error code 22 will be placed in the FILE-STATUS name, if one exists.

Rules for Random and Dynamic Access

1. Random and dynamic access modes require that the primary key of the record to be deleted be placed in the RECORD KEY field.

2. If that record does not exist in the file, the INVALID KEY statement is executed and the FILE STATUS field, if any, has a value of 23. INVALID KEY and FILE STATUS are discussed in INDEXED FILE CONCEPTS at the start of this chapter.

OPEN

Function

The OPEN statement initiates the processing of files. It also performs checking of labels and other input-output operations.

Format

```
OPEN { INPUT file-name-1 [, file-name-2] ...
     { OUTPUT file-name-3 [, file-name-4] ...  } ...
     { I-O file-name-5 [, file-name-6] ...
```

General Rules

1. A file opened as INPUT can be accessed only in a READ or START statement.

2. A file opened as OUTPUT can be accessed only in a WRITE statement.

3. A file opened as I-O can be accessed by a READ, WRITE, REWRITE, START, or DELETE statement.

> **Note**
>
> Not all I-O statements can be used in all access modes. Table A-6 in Appendix A specifies the types of I-O statements that are permissible with the different access modes.

4. Following the initial execution of an OPEN statement for a file, the file cannot be opened again until it has been closed.

5. Execution of the OPEN statement does not obtain or release the first data record.

6. When the INPUT or I-O phrase is specified, the execution of the OPEN statement causes the labels to be checked. Output files must be labeled with CREATK before OPEN.

7. The file-description-entry for a file opened in any mode must be equivalent to that used when the file's template was created with CREATK.

8. OPEN OUTPUT does not create an indexed file. It merely opens an existing file for writing. The file template must be created with the MIDAS command CREATK.

READ

Function

For sequential access, the READ statement makes available the next logical record from a file. For random access, the READ statement makes available a record with a specified key.

Format 1 (Sequential or Dynamic)

READ file-name [NEXT] RECORD [INTO data-name-1]

[; AT END imperative-statement]

Format 2 (Random or Dynamic)

READ file-name RECORD [INTO data-name-1]

[; KEY IS data-name-2]

[; INVALID KEY imperative-statement]

General Rules

1. The INTO phrase must not be used when the input file contains logical records of various sizes. The storage area associated with data-name-1 and the record area associated with file-name may be the same storage area.

2. The key name (data-name-2) must be the name of a data item specified as a record key associated with file-name.

3. The key name may be qualified. It may not be subscripted or indexed.

4. The INVALID KEY phrase (Format 2) or the AT END phrase (Format 1) should be specified if no applicable USE procedure is specified for file-name.

5. The associated file must be open in the INPUT or I-O mode at the time this statement is executed.

6. The execution of the READ statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. FILE STATUS is discussed with INDEXED FILE CONCEPTS at the beginning of this chapter.

7. If the INTO phrase is specified, the record being read is moved from the record area to data-name-1 according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with data-name-1 is evaluated after the record has been read and immediately before it is moved to the data item.

   When the INTO phrase is used, the record being read is available in both the input record area and data-name-1.

8. Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined. For indexed files the key of reference is also undefined.

Rules for Format 1 (Sequential and Dynamic Access)

1. Format 1 must be used for all files in sequential access mode.

2. The NEXT phrase must be specified for files in dynamic access mode, when records are to be retrieved sequentially.

3. The record to be made available by a Format-1 READ statement is determined as follows:

   - If the current record pointer was positioned by the START or OPEN statement, the record to which it points is made available, provided that it is still accessible. If the record is no longer accessible, which may have been caused by the deletion of the record or a change in an alternate record key, the current record pointer is updated to point to the next existing record within the established key of reference. That record is then made available.

   - If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file with the established key of reference and then that record is made available.

4. If, at the time of execution of a Format-1 READ statement, the position of the current record pointer for that file is undefined, the execution of that READ statement is unsuccessful.

5. If, at the time of the execution of a Format-1 READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the READ statement is considered unsuccessful.

6. When the AT END condition is recognized, the following actions are taken in the listed order:

   - A value of 10 is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition.
   - If the AT END phrase is specified, control is transferred to the associated imperative statement. Any USE procedure specified for this file is not executed.
   - If the AT END phrase is not specified, then the USE procedure specified for this file is executed, or else execution is aborted.

7. When the AT END condition has been recognized, a Format-1 READ statement for that file must not be executed without first executing one of the following:

   - A successful CLOSE statement followed by the execution of a successful OPEN statement for that file.
   - A successful START statement for that file.
   - A successful Format-2 READ statement for that file.

8. For a file for which dynamic access mode is specified, a Format-1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from that file as described in General Rule 3 for Format 1.

9. If an alternate record key is the key of reference and duplicates are allowed, records having the same value in that key are read in the same order in which they were written.

Rules for Format 2 (Random and Dynamic Access)

1. Format 2 is used for files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.

2. For an indexed file, if the KEY phrase is specified in a Format-2 READ statement, data-name-2 is established as the key of reference for this retrieval.

3. If the KEY phrase is not specified in a Format-2 READ statement, the primary record key is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format-1 READ statements for the file until a different key of reference is established for the file.

4. Execution of a Format-2 READ statement causes the value of the key of reference to be compared with the value contained in the corresponding index until the record having an equal value is found or, for a secondary key, the first record having the value is found. The current record pointer is positioned to this record which is then made available. If no record can be so identified, the INVALID KEY condition exists and execution of the READ statement is unsuccessful.

REWRITE

Function

The REWRITE statement logically replaces a record on a disk file.

Format

REWRITE record-name [FROM data-name]

[; INVALID KEY imperative-statement]

Syntax Rules

1. The record-name and the data-name may refer to the same storage area.

2. The record-name is the name of a logical record in the FILE section of the DATA division and may be qualified.

3. The INVALID KEY phrase must be specified in the REWRITE statement for files for which an appropriate USE procedure is not specified.

General Rules

1. A record must have been read successfully prior to the REWRITE. This is required to lock the record and ensure that it cannot be updated by another program running concurrently.

2. The REWRITE statement can change any or all data-fields in the record except the primary record key. If the primary key last read is changed, control is passed to the INVALID KEY or USE statement, and the value 22 is placed in the FILE STATUS name, if one exists.

3. The file must be opened for I-O for all access methods.

4. The FROM option allows the record to be created in another area. It is equivalent to MOVE data-name TO record-name prior to the execution of the REWRITE statement. The primary key value must equal the key from the previous READ or the INVALID KEY conditions will occur.

5. The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.

6. Prime extension: the logical record released by a successful execution of the REWRITE statement is still available in the record area.

   If the associated file is named in a SAME RECORD AREA clause, the logical record is also available to the program as a record of other files appearing in the same clause.

7. The execution of the REWRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated.

8. The contents of the alternate record key data item of the record being rewritten may differ from those in the record being replaced if DUPLICATES has been used in the MIDAS template; otherwise, the secondary indexes are updated but the INVALID KEY condition is invoked and the record is unchanged.

9. The INVALID KEY condition exists under any of the following conditions:

   - The value contained in the primary record key data item of the record to be replaced is not equal to the value of the primary record key of the last record read from this file.
   - The value contained in an alternate record key data item for which a DUPLICATES clause has not been specified with CREATK is equal to that of a record already stored in the file.

   The updating operation does not take place and the data in the record area is unaffected.

SEEK - PRIME EXTENSION

Function

SEEK is supported syntactically only for compatibility with other COBOL implementations.

Format

SEEK file-name RECORD

General Rules

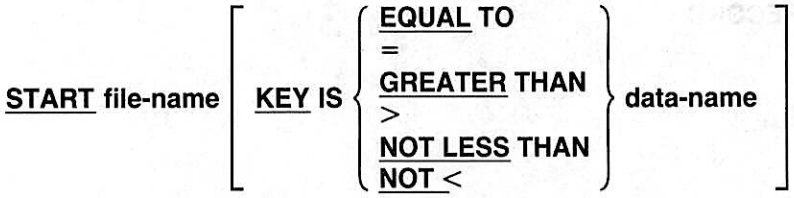
1. SEEK is treated as documentation in Prime COBOL 74.
2. The file-name must be defined by a file-description-entry in the DATA division.

START

Function

The START statement establishes a position in the file for subsequent READs.

Format



Syntax Rules

1. The file-name must be the name of an indexed file with sequential or dynamic access.

2. The data-name may be qualified but not indexed or subscripted. It may reference a data item that is a record key associated with the file.

3. The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.

4. The data-name may reference any data item subordinate to the data-name specified as a record key or alternate record key of file-name, providing its leftmost character position corresponds to the leftmost character position of that key. Thus, partial keys are allowed with this statement.

General Rules

1. File-name must be open in the INPUT or I-O mode at the time that the START statement is executed.

2. If the KEY phrase is not specified, the relational operator 'IS EQUAL TO' and the primary record key are implied.

3. The current record pointer is positioned to the first logical 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 execution of the START statement is unsuccessful, and the position of the current record pointer is undefined.

4. The execution of the START statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. FILE STATUS is explained at the beginning of this chapter.

5. If the KEY phrase is specified, the comparison uses the data item referenced by data-name.

6. After successful execution of the START statement, a key of reference is established and used in subsequent Format-1 READ statements as follows:

   - If the KEY phrase is not specified, the primary record key specified for file-name becomes the key of reference.

   - If the KEY phrase is specified, and data-name is specified as a record key for file-name, that record key becomes the key of reference.

7. START does not retrieve a record, but only positions to a desired record.

8. If execution of START is unsuccessful, the key of reference is undefined.

Example

In the following indexed file, each record contains a NAME field that serves as primary key and a COMPANY field:

data-name	NAME	COMPANY
Picture	PIC X(10)	PIC X(25)
Values	BLYE	REPORTCO
	CLAPP	MERGANTHALER
	FIELDS	SERVICE
	GRIER	AUTOMATION
	HARPER	DESIGNERS
	KEANE	REPORTCO

Source coding to describe the file might be:

```
    ENVIRONMENT DIVISION.
        .
        .
        .
         SELECT FILE-1 ASSIGN TO PFMS
              ORGANIZATION IS INDEXED
              ACCESS IS DYNAMIC
              RECORD KEY IS NAME.
        .
        .
        .
    DATA DIVISION.
    FILE SECTION.
    FD  FILE-1 LABEL RECORDS ARE STANDARD
               VALUE OF FILE-ID IS 'FILE-1'.
    01  FILE-1-RECORD.
        05  NAME      PIC X(10).
        05  COMPANY   PIC X(25).
```

To print records of people whose name begins with the characters F, G, H, and I, program actions should include a START statement to position the file to the first name beginning with one of these letters, and a series of executions of sequential READ statements.

To position with the START statement, the key field (NAME) must first be initialized.

```
    MOVE 'F          ' to NAME.                 Initialize key field.

    START FILE-1 KEY IS NOT LESS THAN NAME      Find the first record
        INVALID KEY STOP RUN.                   whose key is not less
        .                                       than 'F'. This posi-
        .                                       tions the file to
        .                                       this record (FIELDS).

    READ FILE-1 NEXT RECORD,                    Retrieve the first
        AT END STOP RUN.                        record (FIELDS).
        .
        .
    PERFORM 120-READ-NEXT UNTIL NAME NOT        This action will
      LESS THAN 'K'.                            retrieve the records
        .                                       from GRIER through
        .                                       KEANE, and print all
        .                                       except KEANE.
        .

120-READ-NEXT.
    WRITE PRINT-LINE FROM FILE-1-RECORD.
    READ FILE-1 NEXT RECORD
    AT END STOP RUN.
        .
        .
```

USE

USE and the DECLARATIVES header are discussed in Chapter 8.

WRITE

Function

The WRITE statement releases a logical record for an output or input-output file.

Format

WRITE record-name [FROM data-name-1]

[; INVALID KEY imperative-statement]

Syntax Rules

1. The record-name and data-name-1 may name the same storage area.
2. The record-name is the name of a logical record in the FILE section of the DATA division and may be qualified.
3. The INVALID KEY phrase must be specified if an applicable USE procedure is not specified for the associated file.

General Rules

1. The associated file must be open in the OUTPUT or I-O mode.
2. Prime extension: after a successful WRITE, the information is still available in record-name.

   The logical record released by the WRITE statement is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file.

3. Execution of the WRITE statement with the FROM phrase is equivalent to the statement MOVE data-name-1 TO record-name, followed by a WRITE statement.

   After execution of the WRITE statement is complete, the information in the area referenced by data-name-1 is available.

4. The execution of the WRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated. FILE STATUS is explained with INDEXED FILE CONCEPTS at the start of this chapter.

5. The maximum record size for a file is established at the time the file's template is created. The record-name and data-name-1 should not be larger than the record size established with CREATK.

Rules for Record Keys

1. MIDAS or MIDAS stores the written record in such a way that subsequent access may be made based upon any of the record keys.

2. The value of the primary record key must be unique within the records in the file.

3. The data item specified as the primary record key must be set by the program to the desired value prior to the execution of the WRITE statement.

4. Prime extension: in all access modes, records may be written in any order.

5. The value of the alternate record key, if any, may be nonunique only if the DUPLICATES phrase is specified in CREATK for that data item. In this case, the order of retrieval of records with duplicate keys will be the order in which they are written.

6. The INVALID KEY condition exists under any of the following circumstances:

   - When the value of the primary record key is equal to the value of a primary record key of a record already existing in the file.

   - When 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.

EXAMPLE

This sample program illustrates use of the SAME RECORD AREA clause as well as indexed concepts. Since TRANS-FILE and MASTER-FILE share the same record area, no MOVE or WRITE FROM is necessary to write a transaction record to MASTER-FILE.

The MIDAS routines needed to create the two indexed files are listed in Appendix E.

```
 ID DIVISION.
 PROGRAM-ID.    RANDOM1.
*****************************************************************
 REMARKS. THIS PROGRAM ILLUSTRATES READ, WRITE, REWRITE, AND
     DELETE IN RANDOM ACCESS MODE, AND CREATION OF AN INDEXED
     FILE.

     IN ALL CASES, THE KEY IS IN MASTER-RECORD AS SOON AS
     TRANS-FILE IS READ BECAUSE OF THE SAME AREA CLAUSE.
     NO MOVE OF ACCT-ENTRY TO ACCT-MS IS NECESSARY.
*****************************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. PRIME.
 OBJECT-COMPUTER. PRIME.
 INPUT-OUTPUT SECTION.
*
 FILE-CONTROL.
*
     SELECT MASTER-FILE ASSIGN TO PFMS
         ORGANIZATION IS INDEXED,
         ACCESS MODE IS RANDOM,
         RECORD KEY IS ACCT-MS
         FILE STATUS IS FS-MS.
*
     SELECT TRANS-FILE ASSIGN TO PFMS,
         ORGANIZATION IS SEQUENTIAL,
         FILE STATUS IS FS-TR.
*
     SELECT NEW-FILE ASSIGN TO PFMS,
         ORGANIZATION IS INDEXED,
         ACCESS IS RANDOM,
         RECORD KEY IS ACCT-NEW,
         FILE STATUS IS FS-NEW.
*
     SELECT PRINT-FILE ASSIGN TO PRINTER.
*
 I-O-CONTROL.
     SAME RECORD AREA FOR TRANS-FILE, MASTER-FILE.
```

```
*************************************************************
 DATA DIVISION.
 FILE SECTION.
*
 FD  MASTER-FILE COMPRESSED,
     LABEL RECORDS ARE STANDARD,
     VALUE OF FILE-ID IS KDISBURS,
     RECORD CONTAINS 42,
     DATA RECORD IS MASTER-RECORD.
 01  MASTER-RECORD.
     05 ACCT-MS                    PIC X(3).
     05 DATE-MS                    PIC 9(6).
     05 FILLER                     PIC X(3).
     05 VENDOR-MS                  PIC X(20).
     05 CHECK-MS                   PIC X(3).
     05 AMT-MS                     PIC 9(7).
*
 FD  TRANS-FILE COMPRESSED,
     LABEL RECORDS ARE STANDARD,
     VALUE OF FILE-ID IS TRANSFL,
     RECORD CONTAINS 43,
     DATA RECORD IS TRANS-RECORD.
 01  TRANS-RECORD.
     05 TRANS-ENTRY.
        10 ACCT-ENTRY              PIC X(3).
        10 FILLER                  PIC X(39).
     05  ENTRY-CODE                PIC X.
*
 FD  NEW-FILE COMPRESSED,
     LABEL RECORDS ARE STANDARD,
     VALUE OF FILE-ID IS NEWFILE,
     RECORD CONTAINS 42,
     DATA RECORD IS NEW-RECORD.
 01  NEW-RECORD.
     05 NEW-ENTRY.
        10 ACCT-NEW                PIC X(3).
        10 FILLER                  PIC X(38).
     05 NEW-CODE                   PIC X.
*
 FD  PRINT-FILE,
     LABEL RECORDS ARE OMITTED,
     RECORD CONTAINS 42,
     DATA RECORD IS PRINT-LINE.
 01  PRINT-LINE                    PIC X(42).
*
 WORKING-STORAGE SECTION.
 77  FS-NEW                        PIC XX VALUE '00'.
 77  FS-MS                         PIC 99 VALUE 00.
 77  FS-TR                         PIC XX VALUE '00'.
 77  KDISBURS                      PIC X(28)
                 VALUE 'ANNE.K PASSWD>MIDAS>KDISBURS'.
 77  NEWFILE                       PIC X(27)
                 VALUE 'ANNE.K PASSWD>MIDAS>NEWFILE'.
 77  NO-MORE-INPUT                 PIC X VALUE 'N'.
```

```
 77  PRINT-COUNT                    PIC 99 VALUE 00.
 77  TRANSFL                       PIC X(28)
                  VALUE 'ANNE.K PASSWD>MIDAS>TRANSFL'.
*****************************************************
 PROCEDURE DIVISION.
*
*DECLARATIVES.
*    THIS SECTION SHOULD DISPLAY FILE-STATUS FOR ANY ERRORS
*   NOT CAUGHT BY INVALID KEY OR AT END CLAUSES.
*END DECLARATIVES.
*
 000-MAINLINE.
     READY TRACE.
     OPEN INPUT TRANS-FILE,
          I-O MASTER-FILE,
          I-O NEW-FILE,
          OUTPUT PRINT-FILE.
     PERFORM 010-PRINT-HEADINGS.
     READ TRANS-FILE AT END
          DISPLAY 'INPUT FILE IS EMPTY',
          CLOSE TRANS-FILE, MASTER-FILE, NEW-FILE, PRINT-FILE,
          STOP RUN.
     PERFORM 020-PROCESS-TRANS UNTIL NO-MORE-INPUT = 'Y'.
     CLOSE TRANS-FILE,
         MASTER-FILE,
         NEW-FILE,
         PRINT-FILE.
     STOP RUN.
*
 010-PRINT-HEADINGS.
*NOT INCLUDED.
*
 020-PROCESS-TRANS.
     IF ENTRY-CODE = 'U' PERFORM 100-UPDATE
        ELSE IF ENTRY-CODE = 'A' PERFORM 110-ADD
             ELSE IF ENTRY-CODE = 'D' PERFORM 120-DELETE
                  ELSE PERFORM 200-CREATE-ERROR-FILE.
     READ TRANS-FILE AT END
         MOVE 'Y' TO NO-MORE-INPUT
         DISPLAY 'END OF FILE',
         IF PRINT-COUNT = 0 MOVE 'NO PRINT RECORDS' TO PRINT-LINE,
           WRITE PRINT-LINE AFTER ADVANCING 2.
*
 100-UPDATE.
     READ MASTER-FILE INVALID KEY
         MOVE 'N' TO NEW-CODE
         PERFORM 200-CREATE-ERROR-FILE.
     REWRITE MASTER-RECORD INVALID KEY
         DISPLAY 'INVALID REWRITE'.
*
 110-ADD.
     WRITE MASTER-RECORD INVALID KEY
        DISPLAY 'I GOT HERE', MOVE 'D' TO ENTRY-CODE,
```

```
            PERFORM 200-CREATE-ERROR-FILE.
      *
       120-DELETE.
           DELETE MASTER-FILE RECORD, INVALID KEY
               MOVE 'N' TO ENTRY-CODE,
               PERFORM 200-CREATE-ERROR-FILE.
      *
       200-CREATE-ERROR-FILE.
      *        ERRONEOUS INPUT RECORDS ARE WRITTEN TO THE INDEXED
      *        NEW-FILE UNLESS A KEY IS DUPLICATED WITHIN NEW FILE.
      *        IN THAT CASE, THE ERROR RECORD IS PRINTED INSTEAD.
           MOVE ENTRY-CODE TO NEW-CODE.
           MOVE TRANS-ENTRY TO NEW-ENTRY.
           WRITE NEW-RECORD, INVALID KEY
               MOVE NEW-RECORD TO PRINT-LINE,
               WRITE PRINT-LINE AFTER ADVANCING 1,
               ADD 1 TO PRINT-COUNT.
```

To compile, load, and execute this file, stored as RANDOM.CBL, use the following dialog.

```
OK, CBL RANDOM -LIST

[CBL rev x.x]
OK, SEG -LOAD
[SEG rev 19.0]
$ LO RANDOM
$ LI CBLLIB
$ LI
LOAD COMPLETE
$ EXEC
trace:  010-PRINT-HEADINGS
trace:  020-PROCESS-TRANS
trace:  110-ADD
trace:  020-PROCESS-TRANS
trace:  120-DELETE
trace:  020-PROCESS-TRANS
trace:  100-UPDATE
trace:  020-PROCESS-TRANS
trace:  200-CREATE-ERROR-FILE
trace:  020-PROCESS-TRANS
trace:  100-UPDATE
trace:  200-CREATE-ERROR-FILE
End of file. (KX$MYB)
    26 MIDAS ERROR
INVALID REWRITE
END OF FILE
OK,
```

Input File

This display results from use of the following input file to KDISBURS:

```
408080178   ASHTABULA HDWE          4300035476
409080178   CAIRO CHEMICAL          4360002746
410080278   ST.BOTOLPHSTOWN SUPP4200005108
411080278   DOVER MUTUAL            4100034166
412080378   PARIS AUTO              4100015000
413090378   ROME BOATING            4150017982
C82080778   ODESSA SERVICES         4100004670
4500B0778   ANTIOCH SERVALL         4300002580
580080778   BETHLEHEM TAXI          RR00009840
681080778   ATHENS LUMBER           18500036BB
```

Output File

The print-file will look like this.

```
NO PRINT RECORDS
```

For subsequent executions, enter SEG RANDOM.

13
Relative Files

FUNCTION OF THE RELATIVE I-O MODULE

The Relative I-O module allows access of records of a disk file in either a random or a sequential manner. Each record in a relative file is uniquely identified by its relative key, an integer value that specifies the record's position in the file.

The Prime COBOL Relative I-O module is intended to be used with direct access (relative) files created with the MIDAS or MIDASPLUS utility. The COBOL relative key corresponds to the MIDASPLUS direct access primary index. The relative key is part of the MIDASPLUS record description but is not part of the COBOL record description. How to prepare files for COBOL access with MIDASPLUS is discussed in Appendix E. The subject is presented in more detail in the section on DIRECT ACCESS in the MIDAS User's Guide.

> **Caution**
>
> Do not use COBOL with an outdated revision of MIDAS or MIDASPLUS.

LOADING AND EXECUTING PROGRAMS WITH RELATIVE FILES

Use the SEG command steps in Chapter 3 to load a runfile. An example of loading is given at the end of this chapter.

RELATIVE FILE CONCEPTS

Organization

Relative file organization is permitted only on disk. A relative file consists of records that are identified by relative record numbers. The file may be thought of as composed of a serial string or array of areas, each capable of holding a logical record. Each of these areas is identified by a relative record number. Records are stored and retrieved based on this number. For example, the tenth record is the one addressed by relative record number 10 and is in the tenth record area, whether or not records have been written in the first through the ninth record areas.

The file may be represented in Figure 13-1.

No.	Record
1	JAN 40102
2	FEB 29800
3	MAR 45895
4	
5	
6	
7	
8	
9	SEPT 7921
10	OCT 7580
11	NOV 8400
12	DEC 10298

A Relative File
Figure 13-1

The data item named in the RELATIVE KEY clause of the file-control-entry for a file contains the current record number for that file. For inserting, updating, and deleting records in a file, each record is identified solely by the value of its relative key. This value must, therefore, be unique and must not be changed when updating the record. The RELATIVE KEY data item is not part of the file's COBOL record description.

A file used as a relative file in a program must have a template created as a direct access file with MIDASPLUS. The maximum number of records must have been set and all fields must have been declared when the file template was created; they cannot be changed merely by their description in the COBOL program.

Access Modes

Three access modes are possible in COBOL for relative files. They are specified in the SELECT clause:

- In sequential access mode, the sequence in which records are accessed is the ascending order of the relative key values.
- In random access mode, the sequence in which records are accessed is controlled by the program. The desired record is accessed by placing its relative record number in the relative key data item.
- In dynamic access mode, the programmer may change at will from sequential access to random access for reading the file. Access mode requirements for each statement are discussed in the section COMMON OPERATIONS ON RELATIVE FILES later in this chapter.

Current Record Pointer

The current record pointer is a conceptual entity used to indicate the next record to be accessed within a given file. The concept of the current record pointer has no meaning for a file opened in the output mode. The setting of the current record pointer is affected only by the OPEN, START, and READ statements.

File Status

A file status check should be coded in the program to determine the success or failure of an I-O operation. The result can be used to control the next action. If the FILE STATUS clause is specified in a file-control-entry, a value is automatically placed in the specified data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE,

DELETE, or START statement to indicate the status of that input-output operation. FILE STATUS is discussed below with the ENVIRONMENT division. A list of file status codes is given in Table A-5 of Appendix A.

The INVALID KEY Condition

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

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

1. A value is placed in 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, but a USE procedure is specified, either explicitly or implicitly, for this file, that procedure is executed.

When the INVALID KEY condition occurs, execution of the input-output statement that caused the condition is unsuccessful and the file is not affected by the statement involved in the condition.

The AT END Condition

The AT END condition can occur as a result of the execution of a READ statement. For details of the causes of the condition, see the READ statement.

COMMON OPERATIONS ON RELATIVE FILES

Files must be opened with the OPEN statement before any other I-O statements are executed, and must then be closed with CLOSE before the program ends. A file must be closed before being reopened in another mode of operation.

The concept of relative key is essential for most of the following operations on relative files, since MIDASPLUS recognizes only the keys or indexes, and knows nothing about the order or location of the data records themselves.

Create a File

Use the MIDASPLUS CREATK command to create a direct access file template. Then create records on that template either with the MIDASPLUS KBUILD command and an existing data file, or with a COBOL program. Relative files cannot be created by a COBOL program until a MIDASPLUS template exists.

Create (Add) Records

New records are added with WRITE. In sequential access, the records written begin with the first position regardless of any value in the key field. In random or dynamic access, the proper value must be in the relative key before WRITE. In random or dynamic mode, records may be written in any order and inserted anywhere in an existing file.

Position the File to a Certain Record

For sequential access or for dynamic access on sequential files, START should be used to position the file before the first READ. An exception is a sequential access that starts with the first record in the file. If groups of records within the file are to be read sequentially, more than one START may be used.

In random access mode, the data-name after KEY IS must be used and must contain the position of the record for each READ. START is not used.

Read a Certain Record

If the file is opened in sequential access, to read from the start of the file in sequential order of the key no special operations other than READ are necessary. If the first record is not desired, the first READ should be preceded by START in sequential or dynamic mode to specify the key of the first record sought. All subsequent READs will access each next record in key order until another START is used.

If random access is used, a key must be specified before each READ by moving a value into the KEY field or by START.

If access will be both sequential and random, dynamic access mode should be specified, and READ NEXT should be used to change from random to sequential reading.

Delete a Certain Record

The DELETE statement is used. In all access modes, the proper value must be placed in the key field before the DELETE. In sequential mode, the record must be read first.

Update (Change) a Certain Record

The REWRITE statement is used, and the file must be opened for I-O. Before a rewrite in any access mode, the record must first have been read to assure that it cannot be updated by another program running concurrently.

Handle Errors in All of These Statements

The INVALID KEY clause and the USE statement in the declarative section provide error handling. One of these elements is required for an I-O statement. AT END is used only with READ in sequential mode.

ENVIRONMENT DIVISION

This section stresses information that is unique to relative files. Information for the ENVIRONMENT division for sequential and indexed files is given in Chapters 6 and 12, respectively.

INPUT-OUTPUT SECTION -- FILE CONTROL

Function

The FILE-CONTROL paragraph names each file and specifies other file-related information.

Format

```
SELECT file-name ASSIGN TO PFMS

  ; ORGANIZATION IS RELATIVE

[ ; ACCESS MODE IS { SEQUENTIAL [, RELATIVE KEY IS data-name-1] }
                   { { RANDOM  } , RELATIVE KEY IS data-name-1 }
                   { { DYNAMIC }                               } ]

[ ; FILE STATUS IS data-name-2].
```

General Rules

1. The SELECT clause specifies the name of the relative file. The SELECT clause must be specified first in the file-control-entry. The remaining clauses may appear in any order. All relative files are Prime File Management System (PFMS) disk files.

2. The ORGANIZATION IS RELATIVE clause specifies that the file named in the SELECT clause contains data organized by relative keys, and that it is to be processed by MIDASPLUS. The file organization is established at the time the file is created and cannot be changed by this clause.

3. The ACCESS MODE clause specifies how a file is written or read. If it is omitted, sequential access is implied.

   - When SEQUENTIAL is specified, records will be written or retrieved in the order of ascending record number.

   - When RANDOM is specified, the records are to be written or retrieved randomly only, based on the value placed in the RELATIVE KEY field prior to a READ or WRITE. Random mode precludes a sequential READ or WRITE.

   - When DYNAMIC is specified, a program can read randomly or sequentially, and START may be used. Other operations follow the rules for random access.

4. The RELATIVE KEY clause specifies the data item used for the primary MIDASPLUS index. Data-name-1 is used to communicate a relative record number between the COBOL program and MIDASPLUS. RELATIVE KEY need not be specified for sequential access.

   - Data-name-1 must not be defined in the record description associated with file-name. It must refer to an unsigned integer.

   Note

   Data-name-1 cannot have a value exceeding 999,999. It may be described with DISPLAY or COMP-3 as large as PIC 9(6), or with COMP as large as PIC 9(18). If its value exceeds 999,999, or the maximum value allowed by its PICTURE, the value will be truncated at runtime and results will be unpredictable. The size of the relative key in the COBOL program need not be as large as the size specified for the corresponding index in the MIDASPLUS file. (See Appendix B.)

   - Data-name-1 must not be specified with an OCCURS clause, or be contained within a group subordinate to an OCCURS clause. This means it may not be subscripted or indexed, but it may be qualified.

   - Data-name-1 must not be specified with a P character or a separate sign in its PICTURE clause.

5. All records stored in a relative file are uniquely identified by relative record numbers (the MIDASPLUS primary index). The relative record number of a given record specifies the record's logical ordinal position in the file. The first logical record has a relative record number of 1, and subsequent logical records have relative record numbers of 2, 3, 4, and so on.

6. In the FILE STATUS clause, data-name-2 must be a two-character field described in the DATA division. The file control system moves a value into data-name-2 following the execution of every statement that explicitly or implicitly references the file. This value indicates the execution status of the statement to the program. Following a successful I-O operation, data-name-3 contains '00'. The complete status codes are described in Table A-5 of Appendix A.

   The file status field (data-name-2) may not be part of the record description for its file. It may be qualified but must not be subscripted or indexed.

   Prime extension: data-name-2 (FILE-STATUS) may be either alphanumeric or numeric (PIC XX or PIC 99).

I-O-CONTROL

Function

The I-O-CONTROL paragraph specifies the memory area to be shared by different files.

Format

```
I-O CONTROL.

  [ ; SAME [ RECORD ] AREA FOR file-name-1 {,file-name-2} ... ] ...

  [ ; RERUN [ ON file-name-3]
                                  { integer-1 CLOCK-UNITS    }
        EVERY                     { integer-2 RECORDS OF file-name-4 }  ] ...
                                  { condition-name           }
```

General Rules

1. The SAME RECORD AREA or SAME AREA clause specifies that two or more files are to use the same memory area for processing of the current logical record. This saves memory space and eliminates MOVEs between record areas. A logical record in the SAME RECORD AREA is considered both as a logical record of each opened output file in this clause, and as a record of the most recently read input file in this clause. This is equivalent to an implicit redefinition of the area; that is, records are aligned on the leftmost character position.

2. If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in the first clause must appear in the second clause. However, additional file-names not appearing in that SAME AREA clause may also appear in that SAME RECORD AREA clause.

3. The files referenced in the SAME AREA or SAME RECORD AREA clause need not all have the same organization or access.

4. On Prime systems, SAME AREA is equivalent to SAME RECORD AREA.

5. The RERUN clause specifies points in the program at which processing can be restarted in case of premature end of execution. The clause is syntax-checked only.

DATA DIVISION

The elements of the DATA division for relative files are the same as those described in Chapter 7 except for the following three items.

RECORD-DESCRIPTION-ENTRY

COBOL record lengths and MIDASPLUS record lengths must be the same. The relative-key portion of the record may be described as FILLER in the COBOL description, as in the example at the end of this chapter.

RELATIVE KEY

The relative key data item must not be part of the record description for its file. See INPUT-OUTPUT SECTION -- FILE CONTROL above for a discussion.

PROCEDURE DIVISION

The COBOL statements listed in this chapter are described only as they apply to relative file processing. A complete description of all COBOL verbs is provided in Chapter 8, THE PROCEDURE DIVISION.

CLOSE

Function

The CLOSE statement terminates the processing of files.

Format

CLOSE file-name-1 [, file-name-2] ···

Syntax Rule

The files named in the CLOSE statement need not all have the same organization or access.

General Rule

Once a CLOSE statement has been executed for a file, no other statement can be executed for that file unless an intervening OPEN statement for that file is executed.

DELETE

Function

The DELETE statement removes a record from a disk file.

Format

DELETE file-name RECORD [; INVALID KEY imperative-statement]

Syntax Rules

1. The INVALID KEY clause must not be specified for a DELETE statement on a file open in sequential access mode.

2. The INVALID KEY clause must be specified for a DELETE statement on a file open in relative or dynamic access mode for which a USE procedure is not specified.

General Rule

The DELETE statement logically removes a data record from the file together with the index. The file must be open in I-O mode. Execution of DELETE causes the value of the FILE STATUS data item, if any, to be updated. It does not affect the contents of the record area associated with file-name.

Rules for Sequential Access

In sequential access, the record to be deleted must have been successfully read before a DELETE can be executed. The RELATIVE KEY cannot be changed between the READ and DELETE statements.

Rules for Random and Dynamic Access

Random and dynamic access modes also require that the record first be read. If that record does not exist in the file, the INVALID KEY statement is executed if one exists; otherwise the appropriate USE declarative procedure is invoked. The FILE STATUS field, if any, will contain a value of 23. INVALID KEY and FILE STATUS are discussed in RELATIVE FILE CONCEPTS at the beginning of this chapter.

OPEN

Function

The OPEN statement initiates the processing of files. It also performs checking of labels and other input-output operations.

Format

```
OPEN { INPUT file-name-1 [, file-name-2] ...
     { OUTPUT file-name-3 [, file-name-4] ...  } ...
     { I-O file-name-5 [, file-name-6] ...
```

General Rules

1. A file opened as INPUT can be accessed only in a READ or START statement.

2. A file opened as OUTPUT can be accessed only in a WRITE statement.

3. A file opened as I-O can be accessed only by a READ, WRITE, REWRITE, START, or DELETE statement.

> **Note**
>
> Not all I-O statements can be used in all access modes. Table A-6 in Appendix A specifies the types of I-O statements permissible with the different access modes.

4. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement for that same file must be preceded by a CLOSE statement for the file.

5. Execution of the OPEN statement does not obtain or release the first data record.

6. When the INPUT or I-O phrase is specified, the execution of the OPEN statement causes the labels to be checked. Output files must be labeled with CREATK.

7. The file-description-entry for a file opened in any mode must be equivalent to that used when this file's template was created with CREATK.

8. For relative files being opened with the INPUT or I-O phrase, the file is positioned to the first record in the file.

9. OPEN OUTPUT does not create a relative file. It merely opens an existing file for writing. Output file templates must be created and labeled with CREATK.

READ

Function

For sequential access, the READ statement makes available the next logical record from a file. For random access, the READ statement makes available a record with a specified key value.

Format 1 (Sequential or Dynamic)

READ file-name [NEXT] RECORD [INTO data-name-1]

[; AT END imperative-statement]

Format 2 (Random or Dynamic)

READ file-name RECORD [INTO data-name-1]

[; INVALID KEY imperative-statement]

General Rules

1. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions.

2. The INVALID KEY phrase (Format 2) or the AT END phrase (Format 1) should be specified if no applicable USE procedure is specified for file-name.

3. The associated file must be open in the INPUT or I-O mode at the time this statement is executed.

4. The execution of the READ statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. FILE STATUS is discussed with RELATIVE FILE CONCEPTS at the beginning of this chapter.

5. If the INTO phrase is specified, the record being read is moved from the record area to data-name-1 according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with data-name-1 is evaluated after the record has been read and immediately before it is moved to the data item.

   When the INTO phrase is used, the record being read is available in both the input record area and data-name-1.

6. Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined.

Rules for Format 1 (Sequential and Dynamic Access)

1. Format 1 must be used for all files in sequential access mode.

2. The NEXT phrase must be specified for files in dynamic access mode, when records are to be retrieved sequentially. If NEXT is not used, the relative record number of the record to be retrieved must be placed in the key field.

3. The record to be made available by a Format-1 READ statement is determined as follows:

   - If the current record pointer was positioned by the START or OPEN statement, the record to which it points is made available, provided that it is still accessible. If the record is no longer accessible, which may have been caused by the deletion of the record, the current record pointer is updated to point to the next existing record and that record is then made available.

   - If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file. Then that record is made available.

4. If, at the time of execution of a Format-1 READ statement, the position of the current record pointer for that file is undefined, the execution of that READ statement is unsuccessful.

5. If, at the time of the execution of a Format-1 READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the READ statement is considered unsuccessful.

6. When the AT END condition is recognized, the following actions are taken in the specified order:

   - A value is placed in the FILE STATUS data item, if specified for this file, to indicate an AT END condition.
   - If the AT END phrase is specified in the statement, control is transferred to the AT END imperative statement. Any USE procedure specified for this file is not executed.
   - If the AT END phrase is not specified, then the USE procedure specified for this file is executed. Otherwise execution is aborted.

7. When the AT END condition has been recognized, a Format-1 READ statement for that file must not be executed without first executing one of the following:

   - A successful CLOSE statement followed by the execution of a successful OPEN statement for that file.
   - A successful START statement for that file.
   - A successful Format-2 READ statement for that file.

8. In dynamic access mode, a Format-1 READ statement with the NEXT phrase causes the next logical record to be retrieved from that file, as described in General Rule 3 for Format 1.

9. If the RELATIVE KEY phrase is specified, the execution of a Format-1 READ statement updates the contents of the RELATIVE KEY data item such that it contains the relative record number of the record made available.

Rules for Format 2 (Random and Dynamic Access)

1. Format 2 is used for files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.

2. Execution of a Format-2 READ statement causes the value of the relative key to be compared with the relative position of the stored records in the file, until the record having the corresponding record number is found. The current record pointer is positioned to this record, which is then made available. If no record can be so identified, the INVALID KEY condition exists and execution of the READ statement is unsuccessful.

REWRITE

Function

The REWRITE statement logically replaces a record on a disk file.

Format

REWRITE record-name [FROM data-name]

[; INVALID KEY imperative-statement]

Syntax Rules

1. The record-name is the name of a logical record in the FILE section of the DATA division and may be qualified.

2. The INVALID KEY phrase must be specified in random or dynamic access in the REWRITE statement for files for which an appropriate USE procedure is not specified.

General Rules

1. The REWRITE statement can change all data fields in the record.

2. The file must be opened for I-O for all access methods.

3. A record must have been READ successfully prior to the REWRITE. This is required to lock the record and ensure that it cannot be updated by another program running concurrently.

4. The FROM option allows the record to be created in another area. It is equivalent to MOVE data-name TO record-name prior to the execution of the REWRITE statement.

5. The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.

6. Prime extension: after a successful REWRITE, the information is still available in record-name.

   If the associated file is named in a SAME RECORD AREA clause, the logical record is also available as a record of other files in the same clause.

7. The execution of the REWRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated. File status is explained with RELATIVE FILE CONCEPTS at the beginning of this chapter.

8. For a file accessed in either random or dynamic access mode, the record specified by the contents of the RELATIVE KEY data item is replaced. If the file does not contain the record specified by the key, the INVALID KEY condition exists. The updating operation does not take place and the data in the record area is unaffected.

SEEK - PRIME EXTENSION

Function

SEEK is supported syntactically only for compatibility with other COBOL implementations.

Format

SEEK file-name RECORD

General Rules

1. SEEK is treated as documentation in Prime COBOL 74.
2. The file-name must be defined in a file-description-entry in the DATA division.

START

Function

The START statement establishes a position in the file for subsequent READs.

Format

```
START file-name [ KEY IS { EQUAL TO
                         =
                         GREATER THAN
                         >
                         NOT LESS THAN
                         NOT < } data-name ]

    [; INVALID KEY imperative-statement]
```

Syntax Rules

1. The file-name must be the name of a file with sequential or dynamic access.

2. The data-name may be qualified but not indexed or subscripted.

3. The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.

4. The data-name, if used, must be the name in the RELATIVE KEY clause for this file.

General Rules

1. The file-name must be open in the INPUT or I-O mode at the time that the START statement is executed.

2. If the KEY phrase is not specified, the relational operator IS EQUAL TO is implied.

3. The current record pointer is positioned to the first logical 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 execution of the START statement is unsuccessful, and the position of the current record pointer is undefined.

4. The execution of the START statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. FILE STATUS is explained with RELATIVE FILE CONCEPTS at the beginning of this chapter.

5. Whether or not the KEY phrase is specified, the comparison uses the data item referenced by the RELATIVE KEY data-name.

6. START does not retrieve a record, but only positions to a desired record.

USE

USE and the DECLARATIVES header are discussed in Chapter 8.

WRITE

Function

The WRITE statement releases a logical record for an output file.

Format

WRITE record-name [FROM data-name-1]

[; INVALID KEY imperative-statement]

Syntax Rules

1. The record-name and data-name-1 may name the same storage area.
2. The record-name is the name of a logical record in the FILE section of the DATA division and may be qualified.
3. The INVALID KEY phrase must be specified if an applicable USE procedure is not specified for the associated file.

General Rules

1. The associated file must be open in the OUTPUT or I-O mode.
2. Prime extension: the logical record released by the WRITE statement is still available in the record area.

   If the associated file is named in a SAME RECORD AREA clause, the logical record is also available as a record of other files referenced in the clause.
3. Execution of the WRITE statement with the FROM phrase is equivalent to the statement MOVE data-name-1 TO record-name, followed by a WRITE statement.
4. The execution of the WRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated. FILE STATUS is explained with RELATIVE FILE CONCEPTS at the start of this chapter.
5. The maximum record size for a file is established at the time the file template is created. Therefore, record-name and data-name-1 should not be larger than the record defined to MIDAS or MIDASPLUS with CREATK.

Rules for Record Keys

1. When a file is opened for output mode, records may be placed into the file by one of the following ways:

   - If the access mode is sequential, the WRITE statement will cause a record to be released. The first record will be given a relative record number of 1 and subsequent records released will be given relative record numbers of 2, 3, 4, and so on. If the RELATIVE KEY data item has been specified in the file-control-entry entry for the associated file, the relative record number of the record just released will be placed into the RELATIVE KEY data item.

   - If the access mode is random or dynamic, prior to the execution of the WRITE statement, the value of the RELATIVE KEY data item must be initialized with the relative record number. That record is then released.

2. The INVALID KEY condition exists under either of the following circumstances:

   - When the access mode is random or dynamic, and the RELATIVE KEY data item specifies a record that already exists.

   - When an attempt is made to write beyond the externally defined boundaries of the file, as when the relative key value is larger than the number of records allocated with MIDASPLUS for that file.

EXAMPLE

This program illustrates operations on a relative file in random access mode. The MIDASPLUS routines needed to create the two relative files are listed in Appendix E.

```
       ID DIVISION.
       PROGRAM-ID.    RANDOM2.
      ***************************************************************
       REMARKS. THIS PROGRAM ILLUSTRATES READ, WRITE, REWRITE, AND
           DELETE FOR A RELATIVE FILE IN RANDOM ACCESS. IT READS A
           TRANSACTION FILE CONTAINING UPDATES FOR MONTHLY BUDGETS,
           AND UPDATES A MONTH-FILE WHOSE RELATIVE KEY IS THE NUMBER
           OF THE MONTH.
      ***************************************************************
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
           SOURCE-COMPUTER. PRIME.
           OBJECT-COMPUTER. PRIME.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT MASTER-FILE ASSIGN TO PFMS,
               ORGANIZATION IS RELATIVE,
               ACCESS IS RANDOM,
               RELATIVE KEY IS KEY-MS,
               FILE STATUS IS FS-MS.
      *
           SELECT TRANS-FILE ASSIGN TO PFMS,
               FILE STATUS IS FS-TR.
      *
           SELECT NEW-FILE ASSIGN TO PFMS,
               ORGANIZATION IS RELATIVE,
               ACCESS IS DYNAMIC,
               RELATIVE KEY IS KEY-NEW,
               FILE STATUS IS FS-NF.
      *
           SELECT PRINT-FILE ASSIGN TO PRINTER.
      ***************************************************************
       DATA DIVISION.
      *
       FILE SECTION.
       FD  MASTER-FILE COMPRESSED,
           LABEL RECORDS ARE STANDARD,
           VALUE OF FILE-ID IS KMONTHF.
       01  MASTER-RECORD.
           05 INFORMATION     PIC X(29).
           05 FILLER          PIC X(6).
      *
       FD  TRANS-FILE COMPRESSED,
           LABEL RECORDS ARE STANDARD,
           VALUE OF FILE-ID IS TMONTHF.
```

```
 01  TRANS-RECORD.
     05 TRANS-CODE      PIC X.
     05 TRANS-ENTRY     PIC X(29).
     05 KEY-CODE        PIC 99.
*
 FD  NEW-FILE COMPRESSED,
     LABEL RECORDS ARE STANDARD,
     VALUE OF FILE-ID IS OUTMONTF.
 01  NEW-RECORD        PIC X(35).
*
 FD  PRINT-FILE,
     LABEL RECORDS ARE OMITTED.
 01  PRINT-LINE        PIC X(32).
*
 WORKING-STORAGE SECTION.
 77  FS-MS             PIC XX   VALUE '00'.
 77  FS-TR             PIC XX   VALUE '00'.
 77  FS-NF             PIC XX   VALUE '00'.
 77  KEY-MS            PIC 9(2) VALUE ZEROES.
 77  KEY-NEW           PIC 9(2) VALUE ZEROES.
 77  NO-MORE-INPUT     PIC X    VALUE 'N'.
 77  KMONTHF           PIC X(30) VALUE 'KMONTH'.
 77  TMONTHF           PIC X(30) VALUE 'H.ANNE>TMONTH'.
 77  OUTMONTF          PIC X(30) VALUE 'H.ANNE>OUTMONTH'.
*****************************************************************
 PROCEDURE DIVISION.
*
 DECLARATIVES.
 INPUT-ERROR SECTION. USE AFTER STANDARD ERROR PROCEDURE ON INPUT.
 FIRST-PARAGRAPH.
     DISPLAY '**ERROR: **'.
     EXHIBIT FS-MS, FS-TR.
     CLOSE TRANS-FILE, MASTER-FILE, NEW-FILE, PRINT-FILE.
     STOP RUN.
 OUTPUT-ERROR SECTION. USE AFTER STANDARD ERROR PROCEDURE ON
         OUTPUT.
 SECOND-PARAGRAPH.
     DISPLAY '**ERROR: **'.,
     EXHIBIT FS-NF.
     STOP RUN.
 END DECLARATIVES.
*
 000-MAINLINE.
     READY TRACE.
     PERFORM 050-ACCEPT-FILE-NAMES.
     OPEN INPUT TRANS-FILE,
          I-O MASTER-FILE,
          OUTPUT NEW-FILE,
          OUTPUT PRINT-FILE.
     PERFORM 010-UPDATE-MONTHLY-BUDGETS.
     CLOSE TRANS-FILE, MASTER-FILE, NEW-FILE, PRINT-FILE.
     STOP RUN.
*
```

```
 050-ACCEPT-FILE-NAMES.
     DISPLAY 'ENTER MASTER-FILE -- KMONTH OR OTHER'.
     ACCEPT KMONTHF.
     DISPLAY 'ENTER TRANSACTION FILE -- TMONTH OR OTHER'.
     ACCEPT TMONTHF.
     DISPLAY 'ENTER OUTPUT FILE -- OUTMONTH OR OTHER'.
     ACCEPT OUTMONTF.
*
 010-UPDATE-MONTHLY-BUDGETS.
     READ TRANS-FILE AT END DISPLAY 'INPUT FILE WAS EMPTY',
         CLOSE TRANS-FILE, MASTER-FILE, NEW-FILE, PRINT-FILE,
         STOP RUN.
     PERFORM 020-PROCESS-TRANS UNTIL NO-MORE-INPUT = 'Y'.
 020-PROCESS-TRANS.
     MOVE KEY-CODE TO KEY-MS.
     IF TRANS-CODE = 'U' PERFORM 100-UPDATE
         ELSE IF TRANS-CODE = 'A' PERFORM 110-INSERT
              ELSE IF TRANS-CODE = 'D' PERFORM 120-DELETE
                   ELSE EXHIBIT TRANS-RECORD,
                        PERFORM 200-CREATE-ERROR-FILE.
     READ TRANS-FILE AT END DISPLAY 'END OF FILE',
          MOVE 'Y' TO NO-MORE-INPUT.
*
 100-UPDATE.
     READ MASTER-FILE INVALID KEY PERFORM 200-CREATE-ERROR-FILE.
     IF FS-MS = '00',
         REWRITE MASTER-RECORD FROM TRANS-ENTRY,
         INVALID KEY DISPLAY 'INVALID KEY'.
*
 110-INSERT.
     MOVE KEY-CODE TO KEY-MS.
     MOVE TRANS-ENTRY TO MASTER-RECORD.
     WRITE MASTER-RECORD FROM TRANS-ENTRY,
         INVALID KEY PERFORM 200-CREATE-ERROR-FILE.
*
 120-DELETE.
     READ MASTER-FILE INVALID KEY DISPLAY 'INVALID READ'.
     IF FS-MS = '00',
     DELETE MASTER-FILE RECORD INVALID KEY
         PERFORM 200-CREATE-ERROR-FILE.
*
 200-CREATE-ERROR-FILE.
*        THIS CREATES A RELATIVE FILE. IF THE RECORD ALREADY EXISTS,
*        IT IS WRITTEN TO A PRINT FILE INSTEAD.
     MOVE KEY-CODE TO KEY-NEW.
     EXHIBIT NAMED KEY-NEW.
     EXHIBIT FS-MS.
     WRITE NEW-RECORD INVALID KEY PERFORM 210-PRINT-ERROR.
*
 210-PRINT-ERROR.
     WRITE PRINT-LINE FROM TRANS-RECORD.
```

This program, stored as RELATIVE.CBL, may be compiled, loaded, and executed with the following dialog. The first and fifth input records of the file TMONTH cause the program to perform the error routine. End of file causes MIDASPLUS error 27, but does not abort the program.

```
CBL RELATIVE
[CBL  rev x.x]
OK, SEG -LOAD
[SEG rev x.x]
$ LO RELATIVE
$ LI CBLLIB
$ LI
LOAD COMPLETE
$ EXEC
trace:  050-ACCEPT-FILE-NAMES
ENTER MASTER-FILE -- KMONTH OR OTHER
ANNE.K SAMEDI>MIDASPLUS>KMONTH
ENTER TRANSACTION FILE -- TMONTH OR OTHER
ANNE.K SAMEDI>MIDAS>TMONTH
ENTER OUTPUT FILE -- OUTMONTH OR OTHER
ANNE.K SAMEDI>MIDAS>OUTMONTH
trace:  010-UPDATE-MONTHLY-BUDGETS
trace:  020-PROCESS-TRANS
TRANS-RECORD = XThis is wrong                    01
trace:  200-CREATE-ERROR-FILE
KEY-NEW =      1
FS-MS = 00
trace:  020-PROCESS-TRANS
trace:  120-DELETE
trace:  020-PROCESS-TRANS
trace:  100-UPDATE
trace:  020-PROCESS-TRANS
trace:  110-INSERT
trace:  020-PROCESS-TRANS
trace:  100-UPDATE
trace:  200-CREATE-ERROR-FILE
KEY-NEW =     99
FS-MS = 23
End of file. (KX$RAD)
    27 MIDAS ERROR
trace:  210-PRINT-ERROR
END OF FILE
OK,
```

Input Files

```
OK, SLIST MONTHS
This is the January Record    000001
This is the March Record      000003
This is the May Record        000005
This is the June Record       000006
This is the Aug. Record       000008
```

```
This is the Sept. Record      000009
This is the October Record    000010

OK, SLIST TMONTH
XThis is wrong                  01
D                               08
UThis is the September Record   09
AThis is the February Record    02
UInvalid Key                    99
OK,
```

Output File

The print-file contains only one line:

```
OK, SLIST PRINT-FILE

 UInvalid Key                   99
OK,
```

14
Tape Files

INTRODUCTION

Tape Structure

Prime computers support nine-track tape with parity checking. Multiple files on one tape are not supported.

The amount of data that can be put on a tape depends on the following factors:

- Length of the tape in inches.
- Tape density in bytes per inch (bpi). Prime supports 800 and 1600 bpi.
- The blocking factor. The COBOL program's BLOCK CONTAINS clause of the SELECT statement allows grouping of more than one record or character into a block. The block is then read from or written to the tape at once, saving space. If records are not blocked, each record is followed by an interrecord gap or interblock gap (IRG or IBG) of 1/2 inch minimum. (Exceptions are discussed in the Magnetic Tape User's Guide.) Blocking reduces the proportion of tape used by the IRG's. The size of the block that may be created in blocking is limited only by the maximum size of the tape buffer.

- The tape buffer size. For unblocked records, buffer size is the size of one record. Blocked records may occupy a buffer up to the maximum size. The maximum buffer or block size is listed in Appendix J.

COMPILING, LOADING, AND EXECUTING PROGRAMS THAT USE TAPE

No special options are needed on the compile line when tape files are to be processed. The compile line is:

```
CBL file-name
```

If special features are desired, use the compile options listed in Chapter 2.

For loading tape programs, no special libraries or subroutines are required.

Normal Loading

The source program name should end in .CBL. The normal loading sequence for a source program with this name format is:

```
SEG -LOAD
[SEG rev x.x]
$LO program-name     (it is not necessary to add .BIN)
$LI CBLLIB
$LI
     LOAD COMPLETE
$Q
```

If sort files are to be processed, add the command LI VSRTLI before the final LI. You must assign the tape drive before you execute the program.

To run this program, be sure any tape drive assignments are made as discussed below, then enter:

```
SEG program-name
```

The Older Loading Procedure

An older routine for compiling and loading COBOL 74 programs is necessary if the object file-name does not end in .BIN. This older loading routine sequence is:

```
SEG
[SEG rev. x.x]
#LO  program-name.SEG
$LO B_file-name
$LI CBLLIB
$LI
 LOAD COMPLETE
$Q
```

To load subroutines or other object files, use LO file-name after the main object file (B_file-name) is loaded. If sort files are to be processed, add LI VSRTLI before the final LI, as explained in Chapter 3.

To run a program prepared in this way, enter:

```
SEG program-name
```

Tape Drive Assignments at Execution Time

Tape drives are assigned with the PRIMOS-level command:

```
AS[SIGN] MTx [-ALIAS MTy]
```

The value x must represent a physical tape drive from 0 through 7, and the value y must represent a logical tape drive from 0 through 7. The drivename in the file assignments presented below must correspond to the number after ALIAS, if it is used, or else to the number after AS. See the Magnetic Tape User's Guide, and the example at the end of this chapter.

Note

The tape drive number in the file assignment and in the PRIMOS command ASSIGN must be the same. It is, however, independent of the device-name (MT9) in the SELECT statement.

FILE ASSIGNMENTS FOR TAPE

File Assignments with Normal I-O

Normal I-O requires that all file assignments be done within the COBOL program. Use a literal or a data-name in the VALUE OF FILE-ID clause to give the full tape assignment. If the clause contains a data-name, this field can then be given a value interactively with the ACCEPT statement. This technique is particularly helpful for drive numbers that are not known at compile time. Tape file assignment formats are given below.

The compiler searches for a tape assignment to associate with each FD in the following manner:

- If there is no VALUE OF FILE-ID clause for the FD, the compiler uses the file-name after FD.
- If the VALUE OF FILE-ID clause contains a literal, then all characters (up to 120) of the literal are used.
- If the VALUE OF FILE-ID clause contains a data-name, then the complete tape assignment should be contained in the data-name.

Examples are given in the section on VALUE OF FILE-ID below.

File Assignments with -OLD

If the -OLD option is used for compilation, no EXIT PROGRAM statement is included, and FDs are contained in the runfile, then immediately following the execute command SEG runfilename or EXECUTE a request is displayed for runtime file assignments:

```
ENTER FILE ASSIGNMENTS:
>
```

For files whose names within the program are incomplete (which includes all tape files), do the following. Give the literal from the VALUE OF FILE-ID clause of a FILE DESCRIPTION, followed by an equals sign, the name of the actual file to be associated with the program file-name, and a carriage return. The tape file-name format is given below.

If no VALUE OF FILE-ID clause is given for an FD, the compiler generates names in the series F1, F2, and so on. These names can then be used in file assignments in place of the literal.

The system will display the prompt character > while waiting for more user input. The user should make one entry for each FD whose FILE-ID is to be reassigned. (All tape files should be reassigned.)

When no file assignments remain to be entered, use the slash mark to conclude the session. Execution of the application program will then

begin, using the file assignments as entered. The existence and type of each file are checked when OPEN is executed for that file.

Tape File-name Format

To specify the location of a tape file, you must know the drive, the name of the tape volume, and, for files used as input, the owner-id. The formats for tape file assignments are:

- Normal I-O assignment within program:

```
VALUE OF FILE-ID IS 'drivename, label-type, owner-id,
volume-id'
```

- -OLD runtime assignment:

```
tapefile=drivename, label-type, owner-id, volume-id
```

- Assignment within program for either option:

```
VALUE OF FILE-ID IS data-name
.
.
data-name = 'drivename, label-type, owner-id, volume-id'
```

The elements of these formats have the following properties:

tapefile	The literal after VALUE OF FILE-ID in the COBOL program, if one exists; otherwise F1, F2, and so on, or for normal I-O the file-name.
drivename	$MTx, where x is a drive number from 0 through 7 (0 through 7 if logical drives were assigned as discussed with the ALIAS option of ASSIGN in the Magnetic Tape User's Guide).
label-type	The type of label: N no label information S standard labels
owner-id	A 14-character field. This is called the tape file-id by LABEL. This is not the same as the OWNER field in the LABEL command.
volume-id	A six-character field that is written in the label of the tape being created, or is checked if the tape is being read. This is also called the volume serial name (VSN).

Assignment Examples With -OLD

Suppose that in a COBOL program the following statement existed:

```
FD  TAPE-FILE
    LABEL RECORDS ARE STANDARD,
    VALUE OF FILE-ID IS 'FILE2'.
```

Then an appropriate dialog would be:

```
ENTER FILE ASSIGNMENTS:
>FILE2 = $MT0, S, MYNAME, T1
>/
```

The first response above causes the system to look for magnetic tape drive 0, with a tape mounted that contains a volume-id of T1.

Examples using normal I-O are given with the VALUE OF FILE-ID clause below, and with the sample program at the end of this chapter.

Assignment Error Messages

The following are error messages that may be output by the file assignment routine.

- BAD DELIMITER

No equals sign is found, or the equals sign is in an unexpected position.

- ILLEGAL SPECIFICATION

The name to the right of the equals sign begins with $ but does not have the form $MTx.

- LABEL SPECIFICATION EXPECTED

For mag tape, S or N must be specified.

- MTn # OUT OF RANGE

The magnetic tape unit number must be between 0 and 7.

- NAME BUFFER OVERFLOW

The buffer used to store the pathname of the file is full.

- NAME REQUIRED

There is nothing to the right of the equals sign.

- NAME TOO LONG

The name to the right of the equals sign is greater than 14 characters for owner-id or greater than six characters for volume-id.

- TAPE FILE-ID EXPECTED

No tape file-id (owner-id) was specified in a tape assignment.

- VOLUME SERIAL NUMBERS MISMATCH

The volume-id in the file assignment does not match the volume serial number or file-id on the tape label.

- VSN EXPECTED

The volume serial number (volume-id) is missing in a tape assignment.

MULTIVOLUME TAPE FILES

If a tape file is stored on more than one reel of tape, when the end of the first reel is detected, this message will be displayed:

```
**** MTU x END OF VOLUME; MOUNT NEXT VOLUME.
TYPE 'A' TO ABORT, ELSE CORRECT THE PROBLEM AND TYPE RETURN TO
CONTINUE:
```

Put the tape drive offline, rewind the current tape, mount and load the next tape in the series, and then put the drive online again. Press the carriage return to continue execution.

If the wrong reel is mounted, the following message will be displayed:

```
FILE SECTION NUMBERS MISMATCH
TYPE 'A' TO ABORT, ELSE CORRECT THE PROBLEM AND TYPE RETURN TO
CONTINUE:
```

To correct the problem, mount and load the correct tape, put the drive online, and type S to continue execution.

IDENTIFICATION DIVISION

No special commands are needed.

ENVIRONMENT DIVISION

This section discusses only the statements and clauses that are reserved for tape processing. Other features are presented in Chapter 6.

Format

```
INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT [OPTIONAL] file-name-1 ASSIGN TO { MT7 }
                                        { MT9 }

    [ ; RESERVE integer-1 [ AREA  ] ] .
                          [ AREAS ]

[ I-O-CONTROL.

  RERUN [ON file-name-2]

      EVERY { { { END OF { REEL } } }                    }
            { {          { UNIT } } } OF file-name-3 } ...
            { { integer-1 RECORDS  }                   }
            { { integer-2 CLOCK-UNITS                  }
            { { condition-name                         }                ]
```

General Rules

1. The RESERVE clause is for documentation only. Whether or not it is used, one buffer area will be assigned by the compiler.

2. The RERUN clause specifies points in the program at which processing can be restarted in case of premature end of execution. The clause is for documentation only.

Note

An alternate way to assign tape files with normal I-O in the ENVIRONMENT division is to use:

```
SELECT file-name ASSIGN TO PFMS
  .
  .
VALUE OF FILE-ID IS '$MT0, S, MYFILE, T1'
```

For -OLD, the last line could be VALUE OF FILE-ID IS 'X', and then at runtime the file assignment would be:

```
X = '$MTx, S, owner-id, volume-id'
```

DATA DIVISION

This section discusses only features that are unique to tape. Other DATA division elements are presented in Chapter 7.

Format

```
FILE SECTION.

[ FD file-name [ COMPRESSED ]
               [ UNCOMPRESSED ]

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

    [ ; RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS]

    ; LABEL { RECORD IS   } { STANDARD }
           { RECORDS ARE } { OMITTED  }

    [ ; DATA { RECORD IS   } data-name-3 [, data-name-4] ... ]
             { RECORDS ARE }

    [ ; CODE-SET IS alphabet-name]

    [ ; VALUE OF { { FILE-ID } IS { data-name-1 } } ]
                 { { VOL-ID  }    { literal-1  } }

[record-description-entry] ... ]                                ...

[WORKING-STORAGE SECTION.]
```

(See Chapter 7.)

```
[LINKAGE SECTION.]
```

(See Chapter 9.)

BLOCK CONTAINS

Function

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

Format

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

Syntax Rules

1. The BLOCK CONTAINS clause is optional.
2. The clause can only be used in connection with tape files. PRIMOS disk files do not require blocking for efficient access.

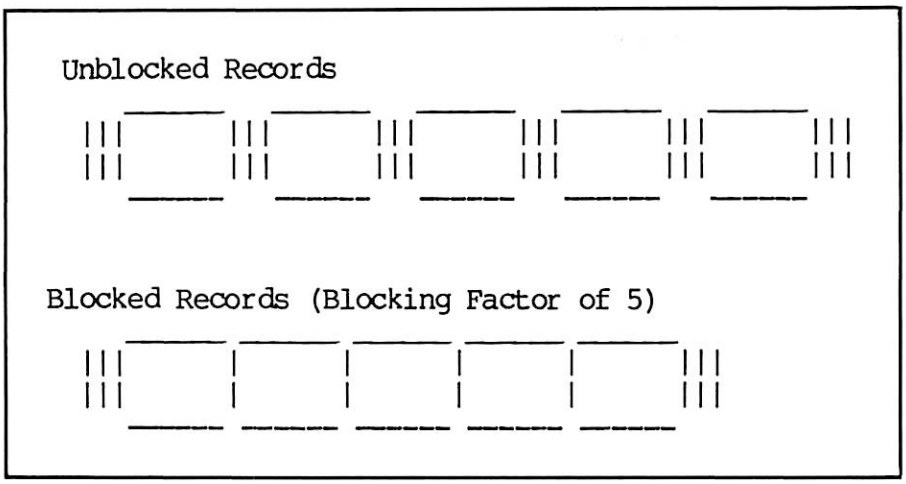
General Rules

1. For an input file, the BLOCK CONTAINS clause must describe the blocking of the records when they were created.
2. The clause may be omitted if the physical record (block) contains one, and only one, complete logical record.
3. If this clause is omitted, records are treated as unblocked.
4. When the RECORDS option is used, the compiler assumes that the block size provides for integer-2 records of the maximum size shown for the file and then provides additional space for any required control words.
5. When the word CHARACTERS is used, the physical record size is specified in terms of the number of character positions required to store the physical record, regardless of the types of characters used to represent the items within the physical record.
6. When neither the CHARACTERS nor the RECORDS option is specified, the CHARACTERS option is assumed.
7. When both integer-1 and integer-2 are used, integer-1 is for documentation purposes only.
8. The maximum size of a block is listed in Appendix J.

Blocking Strategy

Often it is efficient to block records in the largest groups possible. Blocking saves space on the tape because, instead of a 1/2-inch gap after each record, there will be a 1/2-inch gap only after each block of records. Blocking also saves time because only one I-O operation is done per block of records, instead of one operation per record.

The saving of space and, therefore, time is illustrated by Figure 14-1.



Blocking Strategy
(Vertical Lines Represent Interrecord Gap)

Figure 14-1

CODE-SET

Function

The CODE-SET clause specifies the character code set used to represent data on the tape.

Format

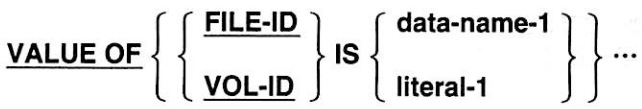
CODE-SET IS alphabet-name

VALUE OF FILE-ID

Function

The VALUE OF FILE-ID clause associates the internal file-name with an external file, thus allowing for the linkage of internal and external program names.

Format



General Rules

1. The literal is an alphanumeric literal that may not exceed 120 characters (eight characters with -OLD).

2. The data-name must be in the WORKING-STORAGE SECTION. It may be qualified, but it must not be subscripted, indexed, or described with USAGE IS INDEX.

3. The VALUE OF FILE-ID clause may be overridden at runtime, if the -OLD compile switch is used, as explained in the section File Assignment, with -OLD earlier in Chapter 14.

4. If there is no VALUE OF FILE-ID clause, a file-name is selected by the compiler according to the processes described in File Assignments with Normal I-O or File Assignments with -OLD above.

5. VOL-ID is reserved for future implementation.

6. For the correct format of the literal or of the contents of the data-name, see Tape File Assignment Format at the start of this chapter.

Examples

A tape file named FILEX can be associated with a logical COBOL file named TEST-FILE in any of the following ways.

1. Value is literal (Normal I-O only):

```
FD TEST-FILE
    LABEL RECORDS STANDARD
    VALUE OF FILE-ID '$MT0, S, MYUFD, FILEX'.
```

At execution time, Normal I-O will simply use FILEX on tape drive 0.

2. Value is data-name:

```
FD TEST-FILE
    LABEL RECORDS STANDARD
    VALUE OF FILE-ID IS TFILE-NAME.
     .
     .
WORKING-STORAGE SECTION.
77 TFILE-NAME                PIC X(20).
```

An actual file-name can be associated with the logical file-name TEST-FILE by executing COBOL statements. For example:

```
IF NEW-FILE = 1
                MOVE '$MT0, S, MYUFD, FILEX' TO TFILE-NAME,
ELSE
                MOVE '$MT1, S, MYUFD, FILEY' TO TFILE-NAME.
```

Another way to do it could be:

```
MOVE SPACES TO TFILE-NAME
DISPLAY "ENTER TEST-FILE NAME."
ACCEPT TFILE-NAME.
```

Then, when the request ENTER TEST-FILE NAME is displayed under either I-O system, enter a name such as $MT0, S, MYUFD, FILEX.

3. With -OLD, if VALUE IS literal is used, the user enters a file assignment at execution time:

```
FD TEST-FILE
    LABEL RECORDS ARE STANDARD
    VALUE OF FILE-ID IS 'MYTAPE'.
.
.
.
>MYTAPE = $MT0, S, MYUFD, FILEX
```

PROCEDURE DIVISION

This section discusses features that are used only for tape processing. A complete presentation of procedural statements is in Chapter 8.

Tape files may be processed only with sequential I-O. Therefore, DELETE, START, REWRITE, and any clauses that are appropriate only for indexed or relative files cannot be used with tape files.

CLOSE

Function

CLOSE terminates the processing of files.

Format

CLOSE file-name-1 [,file-name-2] ...

Syntax Rule

The files referenced in the CLOSE statement need not all have the same organization or access.

General Rules

1. A CLOSE statement implies a preceding OPEN on the same file.

2. If a CLOSE statement has been executed for a file, no other statement can be executed that references that file, unless an intervening OPEN statement for that file is executed.

3. Following the successful execution of a CLOSE statement the record area associated with file-name is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.

4. When standard magnetic tape labels are used, the tape will automatically rewind after a CLOSE statement. With nonstandard labels the tape will stay positioned to the end of the file.

OPEN

Function

OPEN initiates the processing of files and performs checking and writing of labels.

Format

```
OPEN { INPUT file-name-1 [, file-name-2] ···  } ...
     { OUTPUT file-name-3 [, file-name-4] ··· }
```

Syntax Rule

The files referenced in the OPEN statement need not all have the same organization or access.

General Rules

1. The successful execution of an OPEN statement determines the availability of the file, results in the file being in an open mode, and makes the associated record area available to the program. Prior to the successful execution of an OPEN statement for a given file, no statement can be executed that references that file.

2. A file may be opened with the INPUT, OUTPUT, EXTEND, and I-O phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement for that same file must be preceded by the execution of a CLOSE statement for that file.

3. Execution of the OPEN statement does not obtain or release the first data record.

4. If label records are specified for the file, the beginning labels are processed as follows:

   - When the INPUT phrase is specified, the execution of the OPEN statement causes the labels to be checked against the file assignments.

   - When the OUTPUT phrase is specified, the execution of the OPEN statement causes the labels to be written as specified in the file assignments.

5. The file-description-entry for file-name-1, file-name-2, file-name-5, or file-name-6 must be equivalent to that used when this file was created, including blocking of records.

6. For files being opened with the INPUT phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. If no records exist in the file, the current record pointer is set such that the next executed READ statement for the file will result in an AT END condition.

7. Upon successful execution of an OPEN statement with the OUTPUT phrase specified, a file is created. At that time the associated file contains no data records.

READ

Function

READ releases a record from the tape buffer to the program.

Format

READ file-name RECORD [INTO data-name-1]

[; AT END imperative-statement]

The results of a READ from tape are the same as those of a READ from disk. However, you must know how the records were blocked when they were written to tape, and must use the BLOCK CONTAINS clause if necessary to specify the same blocking factor.

To the user, the READ statement appears to read one record at a time from the tape file. However, a whole block is read from tape at once and each subsequent READ statement releases one record to the program.

WRITE

Function

WRITE releases a record to the tape buffer.

Format

WRITE record-name [FROM data-name-1]

The results of a WRITE to tape appear to be the same as those of a WRITE to disk. However, a WRITE statement actually causes a record to be released to the tape buffer until a block is filled. The whole block is then written to the tape.

OVERVIEW OF THE LABEL COMMAND

PRIMOS has a utility called LABEL which initializes magnetic tapes. LABEL writes either IBM (nine-track EBCDIC or seven-track BCD) or ANSI (nine-track ASCII) level-1 volume labels followed by dummy HDR1 and EOF1 labels. LABEL can also be used to read existing VOL1 and HDR1 labels.

ANSI labels are written in accordance with the American National Standards Institute standard ANSI X3.27-1978. IBM labels are written in accordance with IBM's specifications (IBM manual GC28-6680-5).

Any nonstandard labels such as seven-track ASCII or user-defined labels cannot be read or written.

USING LABEL

To read existing labels type the command:

```
LABEL MTn [-TYPE typ]
```

To write labels type the command:

```
LABEL MTn [-TYPE typ] { -VOLUME } vol [-OWNER own] [-ACCESS acc] [-INIT]
                      { -VOLID  }
                      { -VOL    }
```

The arguments have the following meanings:

MTn — Is the tape drive where the tape to be labelled is located. n is a number between 0 and 7. This keyword is required and must be the first on the command line. It matches the logical drive number of the ASSIGN statement.

typ — Is the type of label desired:

-TYPE A	9-track ASCII	(Default)
-TYPE B	7-track BCD	(IBM)
-TYPE E	9-track EBCDIC	(IBM)
-TYPE STANDARD_1 (or S1)	9-track ASCII	(ANSI)

Note

-TYPE STANDARD1 differs from -TYPE A by resetting to zero the leftmost bit of each character in both the label and data areas. This option is useful for transporting tapes between Prime machines and other vendor's machines.

vol Is a one- to 14-character string that uniquely identifies this tape reel. If fewer than 14 characters are specified, they are blank-padded on the right. The keywords VOLUME or VOL may be substituted for the keyword VOLID. This is the volume-id used with the tape assignment.

own Is one to six characters long for ANSI labels, one to ten characters long for IBM labels. If fewer than six characters are specified, they are blank-padded on the right. If this keyword is omitted, the default is the user's login name. The keyword OWN may be substituted for the keyword OWNER.

acc Is a single character defining access to this tape. ACCESS is not used by Prime software but is included for completeness. If it is omitted, it is left blank on ANSI labels. ACCESS is ignored for IBM labels.

INIT Is necessary if the tape is brand new, or if an existing label is to be written over.

On read operations, LABEL prints out the volume-id and owner-id, creation date, access (for ANSI tapes only), and other information.

If LABEL successfully writes a label, the message 'LABEL operation was successful' is displayed.

ERRORS USING LABEL

Improper use of the LABEL command causes an error message to be printed. These errors are the result of bad syntax in the LABEL command itself or of a system magnetic tape I-O error.

Syntax Errors

- Access ignored for IBM labels (warning only).

This is a warning only — processing continues.

- Duplicate keyword detected.

The same keyword was typed more than once.

- Invalid label type specified.

The label type must be one of the characters A, E, or B.

- Invalid tape unit specified.

Something other than MT0 through MT7 was typed.

- Label operation aborted.

One of the four immediately preceding errors occurred and LABEL aborts.

- Label read was not type x.

The label read was not of the type specified.

- No magnetic tape unit specified.

A magnetic tape unit is required.

- Owner ID specified is too long.

The owner-id cannot be longer than 14 characters.

- Owner ID specified is too long for types B or E.

The owner-id for IBM labels cannot be longer than ten characters.

- The tape drive is not ready and/or on-line.

The tape must be mounted on the drive and loaded, then the ONLINE button must be pushed.

- Unable to read tape label on this tape.

A mag tape read error occurred and the label was not read.

- Unable to write tape label on this tape.

A mag tape read error occurred and the label was not written.

- Unrecognized keyword. String (CMDL$A).

An invalid keyword (string) appeared on the command line.

- VOL1 label already exists.

ANSI standards prohibit the rewriting of VOL1 labels. Use the -INIT option.

- Volume ID specified is too long.

The volume-id cannot be longer than six characters.

- Volume ID was not specified.

For writing labels, a volume-id is required.

System Errors

In the following messages, subr is the name of the magnetic tape subroutine that reported the error. See the Subroutines Reference Guide for more information regarding these errors.

Message	Cause
MTn NOT ASSIGNED	Tape drive must be ASSIGNed before using LABEL
subr EOF	END-OF-FILE on the magnetic tape
subr EOT	END-OF-TAPE
subr MTNO	Tape drive is not operational
subr PERR	Parity error on the tape drive
subr HERR	Tape drive hardware error
subr BADC	LABEL improperly called mag tape subroutines

THE HELP FACILITY

The command LABEL -HELP causes LABEL to print out an abbreviated description of the command on the terminal.

For a complete description of tape labels and their use, refer to the IBM publication GC28-6680, OS Tape Labels and the ANSI publication X3.27-1969, American National Standard Magnetic Tape Labels for Information Interchange.

EXAMPLE

The following example is written to run with normal I-O.

```
Source File:  <OPERSY>ANNE.K>NEWCBL>TAPECASH.CBL
Compiled on:  FRI, JUN 11 1982 at 15:29  by: CBL rev 9  06/09/82.09:07:
44.Wed
Options are:  LISTING OPTIMIZE U(PPER)CASE

    1           IDENTIFICATION DIVISION.
    2           PROGRAM-ID.     DISBURSE.
    3           AUTHOR.         ANNE LADD.
    4           INSTALLATION.   PRIME.
    5           DATE-WRITTEN.   SEPTEMBER 20, 1978.
    6           DATE-COMPILED. 820611.15:29:20.
    8           REMARKS.  THIS PROGRAM REPRESENTS THE TAPE PART OF THE
    9               PROGRAM OLDCASH USED AT THE END OF CHAPTERS 5, 6,
   10               7, AND 8.  THIS PART OF THE PROGRAM WRITES TOTALS
   11               BY DEPARTMENT TO TAPE.
   12          *
   13               TO WRITE TAPE RECORDS, ENTER YES FOR TAPE REQUEST.
   14          ************************************************************
   15           ENVIRONMENT DIVISION.
   16           CONFIGURATION SECTION.
   17           SOURCE-COMPUTER.  PRIME.
   18           OBJECT-COMPUTER. PRIME.
   19           INPUT-OUTPUT SECTION.
   20           FILE-CONTROL.
   21               SELECT TAPE-FILE, ASSIGN TO MT9.
   22          ************************************************************
   23           DATA DIVISION.
   24          *
   25           FILE SECTION.
   26          *
   27           FD  TAPE-FILE,
   28               LABEL RECORD IS STANDARD,
   29               BLOCK CONTAINS 4 RECORDS,
   30               VALUE OF FILE-ID IS TAPENAME,
   31               DATA RECORD IS TAPE-LINE.
   32          *
   33           01  TAPE-LINE              PIC X(20).
   34          *
   35           WORKING-STORAGE SECTION.
   36           77  TAPE-CHOICE      PIC XXX VALUE 'NO '.
   37           77  TAPENAME         PIC X(20)
   38                        VALUE '$MT0, S, ANNE, T1'.
   39           77  TOTAL1           PIC S9(9)V99    COMP-3 VALUE ZERO.
   40           77  TOTAL2           PIC S9(9)V99    COMP-3 VALUE ZERO.
   41           77  TOTAL3           PIC S9(9)V99    COMP-3 VALUE ZERO.
   42           77  TOTAL4           PIC S9(9)V99    COMP-3 VALUE ZERO.
   43           77  TOTAL5           PIC S9(9)V99    COMP-3 VALUE ZERO.
   44           77  TOTAL6           PIC S9(9)V99    COMP-3 VALUE ZERO.
   45           77  VARIABLE-MONTH   PIC X(15) VALUE 'THIS MONTH     '.
```

```
46          ********************************************************
47          *       TAPE OUTPUT
48          ********************************************************
49           01  TAPE-HEADER.
50               05 TAPE-MONTH          PIC X(15)       VALUE SPACES.
51               05 FILLER              PIC X(5)        VALUE SPACES.
52           01  SAVE-TAPE.
53               05 SAVE-DATE-TAPE      PIC 9(6).
54               05 SAVE-ACCT-TAPE      PIC XXX.
55               05 SAVE-TOTAL-TAPE     PIC S9(9)V99      COMP-3.
56          ********************************************************
57           PROCEDURE DIVISION.
58          *
59           DECLARATIVES.
60           TAPE SECTION. USE AFTER ERROR PROCEDURE ON TAPE-FILE.
61           FIRST-PARAGRAPH.
62               DISPLAY '**** I-0 ERROR ON TAPE OUTPUT ***'.
63           END DECLARATIVES.
64          *
65           001-BEGIN.
66               READY TRACE.
67               PERFORM 010-GET-JOBINFO.
68               PERFORM 030-PROCESS-DETAIL.
69               PERFORM 050-DEPT-TOTALS.
70               PERFORM 090-PROCESS-TAPE.
71               DISPLAY '     END OF RUN'.
72               STOP RUN.
73          *
74           010-GET-JOBINFO.
75          *NOT INCLUDED.
76          *
77           030-PROCESS-DETAIL.
78          *NOT INCLUDED
79          *
80           050-DEPT-TOTALS.
81          ********************************************************
82          *  MOVE ARBITRARY NUMBERS TO TOTAL1, TOTAL2, ETC.
83          ********************************************************
84               MOVE 11111111 TO TOTAL1.
85               MOVE 22222222 TO TOTAL2.
86               MOVE 33333333 TO TOTAL3.
87               MOVE 44444444 TO TOTAL4.
88               MOVE 55555555 TO TOTAL5.
89               MOVE 66666666 TO TOTAL6.
90          *
91           090-PROCESS-TAPE.
92               DISPLAY 'IS TAPE OUTPUT DESIRED—ENTER YES OR NO'.
93               ACCEPT TAPE-CHOICE.
94               IF TAPE-CHOICE = 'yes' OR
95                  TAPE-CHOICE = 'YES' PERFORM 095-WRITE-TAPE THRU
96                           097-VERIFY-TAPE,
97               ELSE DISPLAY 'NO TAPE'.
```

```
 98          *
 99           095-WRITE-TAPE.
100               OPEN OUTPUT TAPE-FILE.
101               MOVE VARIABLE-MONTH TO TAPE-MONTH.
102               WRITE TAPE-LINE FROM TAPE-HEADER.
103               ACCEPT SAVE-DATE-TAPE FROM DATE.
104               MOVE '100' TO SAVE-ACCT-TAPE.
105               MOVE TOTAL1 TO SAVE-TOTAL-TAPE.
106               WRITE TAPE-LINE FROM SAVE-TAPE  AFTER ADVANCING 1.
107               MOVE '200' TO SAVE-TOTAL-TAPE.
108               MOVE TOTAL2 TO SAVE-TOTAL-TAPE.
109               WRITE TAPE-LINE FROM SAVE-TAPE AFTER ADVANCING 1.
110               MOVE '410' TO SAVE-ACCT-TAPE.
111               MOVE TOTAL3 TO SAVE-TOTAL-TAPE.
112               WRITE TAPE-LINE FROM SAVE-TAPE AFTER ADVANCING 1.
113               MOVE '420' TO SAVE-ACCT-TAPE.
114               MOVE TOTAL4 TO SAVE-TOTAL-TAPE.
115               WRITE TAPE-LINE FROM SAVE-TAPE AFTER ADVANCING 1.
116               MOVE '430' TO SAVE-ACCT-TAPE.
117               MOVE TOTAL5 TO SAVE-TOTAL-TAPE.
118               WRITE TAPE-LINE FROM SAVE-TAPE AFTER ADVANCING 1.
119               MOVE '440' TO SAVE-ACCT-TAPE.
120               MOVE TOTAL6 TO SAVE-TOTAL-TAPE.
121               WRITE TAPE-LINE FROM SAVE-TAPE AFTER ADVANCING 1.
122               CLOSE TAPE-FILE.
123          *
124           097-VERIFY-TAPE.
125               DISPLAY 'FIRST TAPE RECORD - VERIFICATION ONLY'.
126               OPEN INPUT TAPE-FILE.
127               MOVE SPACES TO TAPE-HEADER, SAVE-TAPE.
128               READ TAPE-FILE INTO TAPE-HEADER.
129               READ TAPE-FILE INTO SAVE-TAPE.
130               EXHIBIT SAVE-TOTAL-TAPE.
131               EXHIBIT SAVE-TAPE.
132               CLOSE TAPE-FILE.
133               EXIT.
134          *
```

The following dialog compiles, loads, and executes the program, stored as TAPECASH.CBL. When the tape record SAVE-TAPE is displayed for verification, the part represented by SAVE-TOTAL-TAPE is not displayed because it is declared as COMP-3.

```
CBL TAPECASH -LIST
[CBL rev x]
OK, ASSIGN MT0
Device MT0 assigned.
OK, SEG -LOAD
[SEG rev x.x]
$ LO TAPECASH
$ LI CBLLIB
$ LI
LOAD COMPLETE
$ EXEC
trace:  010-GET-JOBINFO
trace:  030-PROCESS-DETAIL
trace:  050-DEPT-TOTALS
trace:  090-PROCESS-TAPE
IS TAPE OUTPUT DESIRED--ENTER YES OR NO
YES
trace:  095-WRITE-TAPE
trace:  097-VERIFY-TAPE
FIRST TAPE RECORD - VERIFICATION ONLY
SAVE-TOTAL-TAPE =    11111111.00
SAVE-TAPE = 820611100
     END OF RUN
OK,
```

APPENDIXES

A
Reference Tables

The following tables are included in this appendix:

- COBOL Symbols (Table A-1)
- COBOL Reserved Words (Table A-2)
- ASCII Character Set and Collating Sequence (Table A-3)
- EBCDIC Character Set and Collating Sequence (Table A-4)
- File Status Codes (Table A-5)
- Permissible Input/Output Statements After OPEN Options and Access Modes
- Hexadecimal and Decimal Conversion (Table A-7)
- Octal Conversion Table (Table A-8)
- Hexadecimal Addition Table (Table A-9)
- Decimal Data Type (Overpunch Symbols)

Table A-1
COBOL Symbols

Symbol	Functions
PUNCTUATION SYMBOLS -- Used to punctuate program entries	
. period	1. Terminates entries. Usually required. 2. Signifies the decimal point in numeric literals, or the comma in European notation.
, comma	1. Separates operands or clauses in a series. Optional. 2. European notation for the decimal point in numeric literals.
; semicolon	Separates operands or clauses in a series. Optional.
" quotation mark	Encloses nonnumeric literals.
' apostrophe or single quote mark	Encloses nonnumeric literals (PRIME extension).
CODING SYMBOLS -- Directives to the compiler	
* asterisk	Denotes an explanatory comment line when inserted in column 7 of a source program line.
/ slash	Denotes a skip to the top of a new page during a source listing, when coded in column 7 of a source program line.
- hyphen	Denotes a continuation line when coded in column 7 of a source program line.
d letter d	Denotes a line that is treated as a comment line when compiled without the -DEBUG option. When compiled with the -DEBUG option, the line is treated as a normal source statement.

Table A-1 (continued)
COBOL Symbols

Symbol	Functions
SIGN SYMBOLS AND UNARY OPERATORS -- Used in numeric literals and arithmetic expressions.	
+ plus	1. Sign character in the high-order (leftmost) position of a numeric literal.
	2. Unary operator for multiplication by numeric literal +1.
- minus	1. Sign character in the high-order (leftmost) position of a numeric literal.
	2. Unary operator for multiplication by numeric literal -1.
ARITHMETIC SYMBOLS -- Used in arithmetic expressions	
+ plus	Addition
- minus	Subtraction
* asterisk	Multiplication
** double asterisk	Exponentiation
= equal	Assignment
() parentheses	Enclose expressions to control the sequence in which they are evaluated.
CONDITION SYMBOLS -- Used in conditional test expressions	
= equal	Denotes "is equal to".
> greater than	Denotes "is greater than".
< less than	Denotes "is less than".
() parentheses	Enclose expressions to control the sequence in which conditions are evaluated.

Table A-1 (continued)
COBOL Symbols

Symbol	Functions
EDIT SYMBOLS -- Used in picture clauses	
. decimal point (insertion character)	Inserts a decimal point in the indicated position of an edited item.
, comma (insertion character)	Inserts a comma in the indicated position of an edited item. (May be used in conjunction with floating characters.)
$ dollar sign (floating character)	Floats a dollar sign in an edited item so that exactly one dollar sign is inserted immediately to the left of the most significant nonzero digit in any position where the symbol is used.
/ slash (insertion character)	Inserts a slash in the indicated position of an edited item.
* asterisk (replacement character)	Replaces leading zeros with an asterisk. Each asterisk represents a digit position in an edited item.
+ plus - minus (fixed sign control or floating characters)	1. Fixed sign control character in the low-order (rightmost) position of an edited item picture. The symbol does not replace a digit position. 2. Floats a plus or minus character (from left to right) in an edited item, so that exactly one plus or minus sign is developed immediately to the left of the most significant nonzero digit in any position where the symbol has been used.
B (insertion character)	Inserts blanks in the indicated positions of an edited item.
0 zero (insertion character)	Inserts zeros in the indicated positions of an edited item.
Z (replacement character)	Replaces leading zeros with blanks in the indicated positions of an edited item.

Table A-1 (continued)
COBOL Symbols

Symbol	Functions
CR credit (fixed sign control character)	Fixed sign control character in the low-order (rightmost) position of an edited item picture. It occupies two character positions in the edited result.
DB debit (fixed sign-control character)	Fixed sign-control character in the low-order (rightmost) position of an edited item picture. It occupies two character positions in the edited result.
P (decimal scaling character)	Specifies the location of an assumed decimal point when the point is not within the number that appears in the associated data item
V (assumed decimal point)	Positions an assumed decimal point in a in a field.

Table A-2
COBOL Reserved Words

ACCEPT	COMPUTATIONAL	FILE-CONTROL
ACCESS	COMPUTATIONAL-1*	FILE-ID*
ADD	COMPUTATIONAL-2*	FILLER
ADVANCING	COMPUTATIONAL-3*	FIRST
AFTER	COMPUTE	FOOTING**
ALL	CONFIGURATION	FOR
ALPHABETIC	CONSOLE*	FROM
ALSO**	CONTAINS	GIVING
ALTER	COPY	GO
ALTERNATE	CORR	GOBACK*
AND	CORRESPONDING	GREATER
ARE	COUNT	HEADING**
AREA	CURRENCY	HIGH-VALUE
AREAS	DATA	HIGH-VALUES
ASCENDING	DATE	I-O
ASCII*	DATE-COMPILED	I-O-CONTROL
ASSIGN	DATE-WRITTEN	ID*
AT	DAY	IDENTIFICATION
AUTHOR	DEBUGGING**	IF
BEFORE	DECIMAL-POINT	IN
BLANK	DECLARATIVES	INDEX
BLOCK	DELETE	INDEXED
BOTTOM	DELIMITED	INITIAL
BY	DELIMITER	INPUT
CALL	DEPENDING	INPUT-OUTPUT
CANCEL**	DESCENDING	INSPECT
CBL-SUBSCHEMA	DISPLAY	INSTALLATION
CHARACTER	DIVIDE	INTO
CHARACTERS	DIVISION	INVALID
CLOCK-UNITS**	DOWN	IS
CLOSE	DUPLICATES	JUST
CBLSW0*	DYNAMIC	JUSTIFIED
CBLSW1*	EBCDIC	KEY
CBLSW2*	EJECT*	LABEL
CBLSW3*	ELSE	LAST
CBLSW4*	END	LEADING
CBLSW5*	END-OF-PAGE**	LEFT
CBLSW6*	ENTER	LENGTH
CBLSW7*	ENVIRONMENT	LESS
COBOL	EOP**	LINAGE**
CODE	EQUAL	LINAGE-COUNTER**
CODE-SET	ERROR	LINE
COLLATING	EVERY**	LINES
COMMA	EXCEPTION	LINKAGE
COMMON**	EXHIBIT	LOCK**
COMP	EXIT	LOW-VALUE
COMPRESSED*	EXTEND	LOW-VALUES
COMP-1*	EXTERNAL**	MEMORY
COMP-2*	FD	MERGE
COMP-3*	FILE	MODE

Table A-2 (continued)
COBOL Reserved Words

MODULES**	READER*	START
MOVE	READY*	STATUS
MT7*	RECORD	STOP
MT9*	RECORDS	STRING
MULTIPLE**	REDEFINES	SUBTRACT
MULTIPLY	REEL**	SYNC
NAMED*	REFERENCES	SYNCHRONIZED
NATIVE	RELATIVE	TABLE
NEGATIVE	RELEASE	TALLYING
NEXT	REMARKS*	TAPE
NO	REMAINDER	TERMINAL
NOT	REMOVAL**	THAN
NOTE*	RENAMES	THEN*
NUMBER	REPLACING	THROUGH
NUMERIC	RERUN**	THRU
OBJECT-COMPUTER	RESERVE	TIME
OCCURS	RESET	TIMES
OF	RETURN	TO
OFF	REVERSED**	TOP
OFFLINE-PRINT***	REWIND**	TRACE*
OMITTED	REWRITE	TRAILING
ON	RIGHT	UNCOMPRESSED*
OPEN	ROUNDED	UNIT**
OPTIONAL**	RUN	UNSTRING
OR	SAME	UNTIL
ORGANIZATION	SD	UP
OTHERWISE*	SEARCH	UPON
OUTPUT	SECTION	USAGE
OVERFLOW	SECURITY	USE
OWNER*	SEEK*	USING
OWNER-ID*	SEGMENT-LIMIT	VALUE
PAGE*	SELECT	VALUES
PERFORM	SENTENCE	VARYING
PFMS*	SEPARATE	VOL-ID*
PIC	SEQUENCE	WHEN
PICTURE	SEQUENTIAL	WITH
POINTER	SKIP1*	WORDS**
POSITION	SKIP2*	WORKING-STORAGE
POSITIVE	SKIP3*	WRITE
PRINTER*	SET	ZERO
PROCEDURE	SIGN	ZEROES
PROCEDURES	SIZE	ZEROS
PROCEED	SORT	*
PROGRAM	SORT-MERGE	**
PROGRAM-ID	SOURCE-COMPUTER	+
PUNCH*	SPACE	-
QUOTE	SPACES	/
QUOTES	SPECIAL-NAMES	<
RANDOM	STANDARD	=
READ	STANDARD-1	>

Notes to Table A-2

* Prime reserved word
** Not implemented
*** Not available in Rev. 18.4

ASCII CHARACTER SET AND COLLATING SEQUENCE

The Prime COBOL collating sequence in Table A-3 conforms to the American Standard Code for Information Interchange (ASCII) collating sequence. The following table is arranged in ascending value from top to bottom. For nonprinting characters, the control-character combination explained below is included; for printing characters, the punched-card code is given where available.

Following the chart that lists characters in order of value is a chart (page A-14) that shows all characters on one page.

PARITY

ASCII is a seven-bit code set. Since each byte in a Prime machine contains eight bits, the leftmost bit is designated the parity bit, and is set to 1 (ON) by Prime hardware. This is shown on the second chart (page A-14).

Prime uses standard ASCII for communications with devices. The following points are particularly important to Prime usage.

- Output parity is normally transmitted as a zero (space) unless the device requires otherwise, in which case software computes the transmitted parity. Some controllers (such as MLC) may have hardware to assist in parity generation.

- Input parity is ignored by hardware and by standard software. Input drivers are responsible for making the parity bit suit the host software requirements. Some controllers such as MLC may assist in parity error detection.

- The Prime internal standard for the parity bit is 1, that is, '200 added to the octal value.

KEYBOARD INPUT

Nonprinting characters may be entered into text with the logical escape character (^), plus the octal value. For example, typing ^207 will enter one character into the text, and this character will cause the alarm bell to sound on devices with that feature.

Nonprinting characters may also be entered with the CONTROL key plus the character indicated in the rightmost column of the following table.

Table A-3 Part I
ASCII Character Set and Collating Sequence

ASCII Character		Prime Representation Binary	Hex	Octal	Decimal	Control(1)
LOW-VALUES		0000 0000	00	000	0	
NULL	Null (2)	1000 0000	80	200	128	^@
SOH	Start of header	1000 0001	81	201	129	^A
STX	Start of text	1000 0010	82	202	130	^B
ETX	End of text	1000 0011	83	203	131	^C
EOT	End of transmit	1000 0100	84	203	132	^D
ENQ	End of ID	1000 0101	85	205	133	^E
ACK	Acknowledge	1000 0110	86	206	134	^F
BEL	Audible alarm	1000 0111	87	207	135	^G
BS	Backspace	1000 1000	88	210	136	^H
HT	Horizontal tab	1000 1001	89	211	137	^I
LF	Line feed (3)	1000 1010	8A	212	138	^J
VT	Vertical tab	1000 1011	8B	213	139	^K
FF	Form feed	1000 1100	8C	214	140	^L
CR	Carriage (8) return (4)	1000 1101	8D	215	141	^M
SO	RRS (red ribbon shift)	1000 1110	8E	216	142	^N
SI	BRS (black ribbon shift)	1000 1111	8F	217	143	^O
DLE	RCP (relative copy) (5)	1001 0000	90	220	144	^P
DC1	RHT (relative horizontal tab)	1001 0001	91	221	145	^Q
DC2	HLF (half line feed)	1001 0010	92	222	146	^R
DC3	RVT (relative vertical tab) (4)	1001 0011	93	223	147	^S
DC4	HLF (half line feed reverse)	1001 0100	94	224	148	^T
NAK	Negative acknowledgment	1001 0101	95	225	149	^U
SYN	Synchronocity	1001 0110	96	226	150	^V
ETB	End of transmission block	1001 0111	97	227	151	^W
CAN	Cancel	1001 1000	98	230	152	^X
EM	End of medium	1001 1001	99	231	153	^Y
SUB	Substitute	1001 1010	9A	232	154	^Z
ESC	Escape	1001 1011	9B	233	155	^|
FS	File separator	1001 1100	9C	234	156	^
GS	Group separator	1001 1101	9D	235	157	^
RS	Record separator	1001 1110	9E	236	158	^^

Table A-3 Part I (continued)
ASCII Character Set and Collating Sequence

ASCII Character	Prime Representation Binary	Hex	Octal	Decimal	Card Punch(8)
US Unit separator	1001 1111	9F	237	159	^_
(space)	1010 0001	A0	240	160	No Punch
! (exclamation)	1010 0001	A1	241	161	12-8-2
" (2)	1010 0010	A2	242	162	7-8
# (number)(10)	1010 0011	A3	243	163	8-3
$	1010 0100	A4	244	164	11-3-8
%	1010 0101	A5	245	165	
&	1010 0110	A6	246	166	
' (11)	1010 0111	A7	247	167	5-8
(	1010 1000	A8	250	168	12-5-8
)	1010 1001	A9	251	169	11-5-8
*	1010 1010	AA	252	170	11-4-8
+	1010 1011	AB	253	171	12-6-8
, (comma)	1010 1100	AC	254	172	0-3-8
- (minus or hyphen)	1010 1101	AD	255	173	11
. (period)	1010 1110	AE	256	174	12-3-8
/ (slash)	1010 1111	AF	257	175	0-1
0 (zero)	1011 0000	B0	260	176	0
1	1011 0001	B1	261	177	1
2	1011 0010	B2	262	178	2
3	1011 0011	B3	263	179	3
4	1011 0100	B4	264	180	4
5	1011 0101	B5	265	181	5
6	1011 0110	B6	266	182	6
7	1011 0110	B7	267	183	7
8	1011 1000	B8	270	184	8
9	1011 1001	B9	271	185	9
: (colon)	1011 1010	BA	272	186	8-2
; (semicolon)	1011 1011	BB	273	187	11-6-8
<	1011 1100	BC	274	188	12-4-8
=	1011 1101	BD	275	189	6-8
>	1011 1110	BE	276	190	0-6-8
? (12)	1011 1111	BF	277	191	0-7-8
@ (at)	1100 0000	C0	300	192	8-4
A	1100 0001	C1	301	193	12-1
B	1100 0010	C2	302	194	12-2
C	1100 0011	C3	303	195	12-3
D	1100 0100	C4	304	196	12-4
E	1100 0101	C5	305	197	12-5
F	1100 0110	C6	306	198	12-6
G	1100 0111	C7	307	199	12-7
H	1100 1000	C8	310	200	12-8
I	1100 1001	C9	311	201	12-9
J	1100 1010	CA	312	202	11-1
K	1100 1011	CB	313	203	11-2

Table A-3 Part I (continued)
ASCII Character Set and Collating Sequence

ASCII Character	Prime Representation Binary	Hex	Octal	Decimal	Card Punch(8)
L	1100 1100	CC	314	204	11-3
M	1100 1101	CD	315	205	11-4
N	1100 1110	CE	316	206	11-5
O	1100 1111	CF	317	207	11-6
P	1101 0000	D0	320	208	11-7
Q	1101 0001	D1	321	209	11-8
R	1101 0010	D2	322	210	11-9
S	1101 0011	D3	323	211	0-2
T	1101 0100	D4	324	212	0-3
U	1101 0101	D5	325	213	0-4
V	1101 0110	D6	326	214	0-5
W	1101 0111	D7	327	215	0-6
X	1101 1000	D8	330	216	0-7
Y	1101 1001	D9	331	217	0-8
Z	1101 1010	DA	332	218	0-9
[	1101 1011	DB	333	219	
\ (backslash)	1101 1100	DC	334	220	
]	1101 1101	DD	335	221	
^ (Escape) (13)	1101 1110	DE	336	222	
_ (Underscore) (14)	1101 1111	DF	337	223	
` (Grave)	1110 0000	E0	340	224	
a	1110 0001	E1	341	225	
b	1110 0010	E2	342	226	
c	1110 0011	E3	343	227	
d	1110 0100	E4	344	228	
e	1110 0101	E5	345	229	
f	1110 0110	E6	346	230	
g	1110 0111	E7	347	231	
h	1110 1000	E8	350	232	
i	1110 1001	E9	351	233	
j	1110 1010	EA	352	234	
k	1110 1010	EB	353	235	
l	1110 1100	EC	354	236	
m	1110 1101	ED	355	237	
n	1110 1110	EE	356	238	
o	1110 1110	EF	357	239	
p	1111 0000	F0	360	240	
q	1111 0001	F1	361	241	
r	1111 0010	F2	362	242	
s	1111 0011	F3	363	243	
t	1111 0100	F4	364	244	
u	1111 0101	F5	365	245	
v	1111 0110	F6	366	246	
w	1111 0111	F7	367	247	
x	1111 1000	F8	370	248	
y	1111 1001	F9	371	249	

Table A-3 Part I (continued)
ASCII Character Set and Collating Sequence

ASCII Character	Prime Representation Binary	Hex	Octal	Decimal	Card Punch(8)
z	1111 1010	FA	372	250	
{	1111 1011	FB	373	251	12-0
|	1111 1100	FC	374	252	
}	1111 1101	FD	375	253	11-0
~ (ESC)	1111 1110	FE	376	254	
DEL (15)	1111 1111	FF	377	255	

Notes to Table A-3 Part I

1. The symbol ^ here signifies pressing the CONTROL key simultaneously with the character key.
2. FILLER
3. Ignored as terminal input.
4. CR is interpreted as .NL. at the terminal.
5. BREAK at terminal.
6. Next byte specifies number of spaces to insert.
7. Next byte specifies number of lines to insert.
8. Characters with no punched cards code, other than the space, are not supported for punched card entry.
9. Double quote -- erase previous character, unless the user or system default has been changed.
10. Pound sign in British use.
11. Apostrophe or single quote.
12. Kill current line, unless the user or system default has been changed.
13. Logical escape or up arrow.
14. Back arrow on some terminals.
15. HIGH-VALUE (rubout or delete is not recognized from terminal).

Table A-3 Part II
ASCII Character Set and Collating Sequence

		Control Characters			Graphic Characters					
		Column	0	1	2	3	4	5	6	7
		Hex	8	9	A	B	C	D	E	F
Row	Hex	Bits	P000	P001	P010	P011	P100	P101	P110	P111
0	0	0000	NUL	DLE	SP	0	@	P	`	p
1	1	0001	SOH	DC1	!	1	A	Q	a	q
2	2	0010	STX	DC2	"	2	B	R	b	r
3	3	0011	ETX	DC3	#	3	C	S	c	s
4	4	0100	EOT	DC4	$	4	D	T	d	t
5	5	0101	ENQ	NAK	%	5	E	U	e	u
6	6	0110	ACK	SYN	&	6	F	V	f	v
7	7	0111	BEL	ETB	'	7	G	W	g	w
8	8	1000	BS	CAN	(	8	H	X	h	x
9	9	1001	HT	EM	)	9	I	Y	i	y
10	A	1010	LF	SUB	*	:	J	Z	j	z
11	B	1011	VT	ESC	+	;	K	[	k	{
12	C	1100	FF	FS	,	<	L	\	l	|
13	D	1101	CR	GS	-	=	M	]	m	}
14	E	1110	SO	RS	.	>	N	^	n	~
15	F	1111	SI	US	/	?	O	_	o	DEL

Note

ASCII is a seven-bit code set. Since each byte on a Prime machine contains eight bits, the leftmost bit is designated the "parity" bit (P), and is set on ("1") by PRIME hardware. For example, if the user enters the string "IS" using the PRIME editor, the resultant two bytes in memory are C 9 D 3 (hexadecimal) or 1100 1001 1101 0011 (binary).

Table A-4
EBCDIC Character Set and Collating Sequence

COBOL Character Set	EBCDIC Codes Hexa-decimal	EBCDIC Codes Punched Card
. (period)	4B	12-3-8
<	4C	
(	4D	12-5-8
+	4E	12-6-8
$	5B	11-3-8
*	5C	11-4-8
)	5D	11-5-8
;	5E	11-6-8
- (minus or hyphen)	60	11
/	61	0-1
, (comma)	6B	0-3-8
	6C	0-4-8
>	6E	
	6F	0-7-8
' (apostrophe)	7D	5-8
=	7E	6-8
" (quote)	7F	7-8
a	81	
b	82	
c	83	
d	84	
e	85	
f	86	
g	87	
h	88	
i	89	
j	91	
k	92	
l	93	
m	94	
n	95	
o	96	
p	97	
q	98	
r	99	
s	A2	
t	A3	
u	A4	
v	A5	
w	A6	
x	A7	
y	A8	
z	A9	

Table A-4 (continued)
EBCDIC Character Set and Collating Sequence

COBOL Character Set	EBCDIC Codes	
	Hexa-decimal	Punched Card
A	C1	12-1
B	C2	12-2
C	C3	12-3
D	C4	12-4
E	C5	12-5
F	C6	12-6
G	C7	12-7
H	C8	12-8
I	C9	12-9
J	D1	11-1
K	D2	11-2
L	D3	11-3
M	D4	11-4
N	D5	11-5
O	D6	11-6
P	D7	11-7
Q	D8	11-8
R	D9	11-9
S	E2	0-2
T	E3	0-3
U	E4	0-4
V	E5	0-5
W	E6	0-6
X	E7	0-7
Y	E8	0-8
Z	E9	0-9
0	F0	0
1	F1	1
2	F2	2
3	F3	3
4	F4	4
5	F5	5
6	F6	6
7	F7	7
8	F8	8
9	F9	9

Table A-5
File Status Codes

Status Code	Error Type	Interpretation
		Sequential Files
00	NONE	Successful completion of operation.
10	END OF FILE	End of file reached on READ; file pointer positioned past logical end of file.
30	PERMANENT I-O ERROR	Hardware I-O error such as data check, parity error, or transmission error.
34	PERMANENT I-O ERROR	Boundary violation: user has attempted to write beyond the externally defined boundaries of a file. Disk space full.
41	FORMS ERROR	FORMS validation error on a READ.
		Relative Files
00	NONE	Successful completion of operation.
10	END OF FILE	End of file encountered during a READ.
21	INVALID KEY	User has attempted to write beyond the externally defined boundaries of the file (disk space full).
22	INVALID KEY	Record already exists in data subfile; user attempted to add a record with a nonunique record number.
23	INVALID KEY	Record not found; no record found with the specified key value.
24	INVALID KEY	Boundary violation; user has attempted to read or write beyond boundaries preallocated by CREATK.

Table A-5 (continued)
File Status Codes

Status Code	Error Type	Interpretation
30	PERMANENT I-O ERROR	Hardware I-O error: could be a parity error, data check, or transmission error.
90	PRIME-DEFINED	Locked record; attempt to access a record already locked by another user or process.
91	PRIME-DEFINED	Unlocked record; REWRITE attempted without first locking the record with a READ.
94	PRIME-DEFINED	MIDAS concurrency error; another user has deleted the record you were trying to access.
95	PRIME-DEFINED	User has supplied a record size for a RELATIVE file that does not match the record size assigned to the file during template creation.
96	PRIME-DEFINED	Relative record number error; user supplied a record number larger than the number preallocated by CREATK.
98	PRIME-DEFINED	Attempt to do an indexed add to a direct access file.
99	PRIME-DEFINED	System error; possibly serious. Verify that error is not due to a START that encountered a locked record before calling your System Analyst.
	Indexed Files	
00	NONE	Successful completion of operation.
10	END OF FILE	End of file reached on READ operation; file pointer positioned past logical end of file (highest key value).
22	INVALID KEY	Attempt to perform a WRITE that would create a duplicate primary key entry, or changing the primary key on a REWRITE.

Table A-5 (continued)
File Status Codes

Status Code	Error Type	Interpretation
23	INVALID KEY	There is no record in the file with this key value.
30	SYSTEM I-O ERROR	Operation unsuccessful due to an I-O error, such as a data check, parity error, or transmission error.
90	PRIME-DEFINED	Record already locked: another user or process has already locked this record for update.
91	PRIME-DEFINED	Record not locked: a REWRITE operation was attempted without first locking the record via a READ operation.
92	PRIME-DEFINED	Attempt to add a duplicate secondary key value to a secondary index subfile that does not permit duplicates.
93	PRIME-DEFINED	The indexes referred to in the program do not match those defined during MIDASPLUS template creation.
94	PRIME-DEFINED	MIDAS concurrency error: another user has just deleted the record you were trying to access.
95	PRIME-DEFINED	Bad record length supplied: the program has incorrectly specified the record length (data size) of the MIDASPLUS file.
99	PRIME-DEFINED	System error. Verify that the program is not seriously flawed before you call your System Analyst.

Table A-6
Permissible Input/Output Statements
After OPEN Options and Access Modes

(X Means Permitted)

File Organization	File Access Mode	Procedure Statement	OPEN Option in Effect			
			INPUT	OUTPUT	I-O	EXTEND
SEQUENTIAL	SEQUENTIAL	READ	X		X	
		WRITE		X	X	X
		REWRITE			X	
INDEXED	SEQUENTIAL	READ	X		X	
		WRITE		X	X	
		REWRITE			X	
		START	X		X	
		DELETE			X	
	RANDOM	READ	X		X	
		WRITE		X	X	
		REWRITE			X	
		START				
		DELETE			X	
	DYNAMIC	READ	X		X	
		WRITE		X	X	
		REWRITE			X	
		START	X		X	
		DELETE			X	
RELATIVE	SEQUENTIAL	READ	X		X	
		WRITE		X		
		REWRITE			X	
		START	X		X	
		DELETE			X	
	RANDOM	READ	X		X	
		WRITE		X	X	
		REWRITE			X	
		START				
		DELETE			X	
	DYNAMIC	READ	X		X	
		WRITE		X	X	
		REWRITE			X	
		START	X		X	
		DELETE			X	

Table A-7
Hexadecimal and Decimal Conversion

To convert a hexadecimal or octal number to decimal with one of the following charts, locate each digit in the correct column position and add the decimal equivalents of all digits. For example, 6C5 in hex equals 1,536 + 192 + 5 in decimal, or 1733.

To convert from decimal to hex or octal, locate the largest decimal value in the table that is still smaller than the number to be converted. Note the corresponding hex or octal value on the left. Then subtract that value from your number, and repeat. Each time, write down the new digit to the right of the last one. For example, 95 is 80 plus 15, or 5 from the second hex column and F from the third column; 95 decimal equals 5F in hex.

HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49152	C	3072	C	192	C	12
D	53248	D	3328	D	208	D	13
E	57344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15
16^3		16^2		16^1		16^0	

Table A-8
Octal and Decimal Conversion

See the explanation of this chart on the preceding page.

OCT	DEC	OCT	DEC	OCT	DEC	OCT	DEC	OCT	DEC
0	0	0	0	0	0	0	0	0	0
1	4096	1	512	1	64	1	8	1	1
2	8192	2	1024	2	128	2	16	2	2
3	12288	3	1536	3	192	3	24	3	3
4	16384	4	2048	4	256	4	32	4	4
5	20480	5	2560	5	320	5	40	5	5
6	24576	6	3072	6	384	6	48	6	6
7	28672	7	3584	7	448	7	56	7	7
8^4		8^3		8^2		8^1		8^0	

Table A-9
Hexadecimal Addition Table

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Table A-10
Decimal Data Type (Overpunch Symbols)

COBOL operates on five types of decimal data. Table A-10 summarizes the characteristics of each type.

Type	Code	Size of Decimal Digit	Comments
Leading Separate Sign	0	8	A plus sign (+) or a space represents a positive number. Operations generate +. A minus sign (-) represents a negative number.
Trailing Separate Sign	1	8	Same as leading separate sign.
Packed Decimal	3	4	Each four bits represent a digit in binary-coded decimal form. The right-most four bits represent the sign of the entire decimal field: bit values 1100 = plus, bit values 1101 = minus.
Leading Embedded Sign	4		A single character represents a digit and the sign of the field. When more than one character is listed, all will be recognized but only the first will be given in the result field.
Trailing Embedded Sign	5	8	Embedded sign characters have the following meaning:

Digit	Positive	Negative
0	0,+{	-,}
1	1A	J
2	2B	K
3	3C	L
4	4D	M
5	5E	N
6	6F	O
7	7G	P
8	8H	Q
9	9I	R

B
Error Messages

Four categories of error messages may be generated when COBOL programs are compiled and run. These messages will appear on the screen if the program is being compiled and run interactively:

- Compile time error messages
- COBOL runtime error messages
- MIDAS or MIDASPLUS runtime error messages
- PRIMOS error messages

COMPILE TIME ERROR MESSAGES

These messages are displayed on the screen. They are also stored in the file program.CBL.ERROR and inserted after the source listing, if either of these two files is specified at compile time.

The format of compile time error messages is explained under COMPILER OUTPUT in Chapter 2. Examples are:

```
ERROR 262 SEVERITY 1 [OBSERVATION, SEMANTICS]
File NEW-FILE was opened but never accessed.

ERROR 307 SEVERITY 2 LINE 121 COLUMN 15 [WARNING, SEMANTICS]
The initial value for 'FILLER' exceeds the range of values allowed
by the PICTURE or by the default implementation size.
```

```
The initial value may be truncated or unpredictable.

ERROR 28 SEVERITY 3 LINE 64 COLUMN 43 [FATAL, SEMANTICS]
A closing quotation mark must be followed by ,.;) or by ' '.

ERROR 266 SEVERITY 4
The number of fatal errors detected for this compilation exceeds
the current implementation limit of 100.

COMPILATION ABORTED.

ERROR 229  SEVERITY 3  LINE 137<2>  COLUMN 24  [FATAL, SEMANTICS]
"NOFILE" is an undefined data reference.
```

The compile time error messages are self-explanatory. If a level-4 message is encountered, a recompilation after elimination of all other error messages will usually eliminate the level-4 message also. If a level-4 message persists, it will be necessary to consult a Prime System Analyst.

COBOL RUNTIME ERROR MESSAGES

These messages are displayed when, for example, subroutines called by COBOL are unable to do operations such as file I-O. The message, which is self-explanatory, describes the error, the file involved if any, and gives the name of the subroutine. An example of a message from subroutine C$ER is:

```
SEQUENTIAL WRITE TO RANDOM FILE OPENED IN I-O MODE:
FILE-ID:  KMONTH     OWNER-ID:  (OMITTED)     DEVICE:  PFMS

   FATAL RUN-TIME I-O ERROR.  (C$ER)
ER!
```

MIDAS OR MIDASPLUS RUNTIME ERROR MESSAGES

These error messages may be displayed when COBOL programs using indexed or relative files are executed. The format of the error message is:

```
MIDAS FILE SYSTEM ERROR n1, FILE STATUS CODE n2
[message                  ]
FILE-ID: filename  OWNER-ID:owner  DEVICE:  device-name
FATAL RUN-TIME I-O ERROR (subroutine-name)
```

MIDASPLUS errors are explained by number in the MIDAS User's Guide. The file status code is the same used by COBOL and is explained in Appendix A of this manual. An example is:

```
MIDAS FILE SYSTEM ERROR   7, FILE STATUS CODE 23
ATTEMPTED READ FROM UNOPENED FILE:
FILE-ID:KMONTH  OWNER-ID: (OMITTED)  DEVICE: PFMS
FATAL RUN-TIME I-O ERROR (subroutine-name)
```

PRIMOS ERROR MESSAGES

These messages, generated by PRIMOS, the Prime operating system, are explained in the Prime User's Guide. An example is:

```
ERROR:  condition POINTER_FAULT$ raised at 4011(3)1011
```

In this case, a sort program was run without loading VSRTLI.

See also RUNTIME ERROR MESSAGES in Chapter 3.

C
FIPS Levels

As explained in Chapter 2, the COBOL 74 compiler does Federal Information Processing (FIPS) syntax-checking if the appropriate option is included on the compile line. The compiler options for the four FIPS levels are -FIPS1, -FIPS2, -FIPS3, and -FIPS4. Table C-1 shows the ANSI COBOL level permitted for each FIPS level. (The difference between ANSI levels 1 and 2 is marked in the ANSI publication X3.23-1974, American National Standard Programming Language COBOL.) Prime extensions are always flagged by this compiler option.

Error messages are displayed on the screen. They are also stored in the file named program.CBL.ERROR and inserted after the source listing if these files are specified at compile time. The error messages are self-explanatory. Examples are:

- Levels 1 and 2:

  ```
  ERROR 24 SEVERITY 1 LINE 6 COLUMN 8 [OBSERVATION, SEMANTICS]
  DATE-COMPILED not allowed on low and low-intermediate FIPS
  levels.
  ```

- All levels:

  ```
  ERROR 32 SEVERITY 1 LINE 8 COLUMN 8 [OBSERVATION, SEMANTICS]
  REMARKS is a Prime extension.
  ```

Table C-1
FIPS and ANSI Levels

(1 or 2 refers to the ANSI level permitted. A dash means not permitted.)

ANSI COBOL Module	FIPS Level			
	1 Low Level	2 Low Intermediate Level	3 High Intermediate Level	4 High Level
Nucleus	1	1	2	2
Table handling	1	1	2	2
Sequential I-O	1	1	2	2
Relative I-O	-	1	2	2
Indexed I-O	-	-	-	2
Sort-merge	-	-	1	2
Report writer	-	-	-	-
Segmentation	-	1	1	2
Library	-	1	1	2
Debug	-	1	2	2
Interprogram communication	-	1	2	2
Communication	-	-	2	2
Prime extensions	-	-	-	-

D
The Debugger Interface

The symbolic debugger is presented in the Source Level Debugger Guide. This appendix describes the COBOL 74 interface to the Debugger, and restrictions to debugging in COBOL.

OVERVIEW

To use the Debugger with COBOL 74, follow these steps:

1. Compile the program using the -DEBUG option. For example, if the program to be debugged is OLDCASH.CBL, enter:

```
CBL OLDCASH -DEBUG
```

2. Load the program in the usual way:

```
SEG -LOAD
[seg rev 18.x]
$LO OLDCASH
$LI CBLLIB
$LI
LOAD COMPLETE
$Q
```

3. Invoke the Debugger:

   DBG OLDCASH [option-1 [option-2]...]

   If the program was compiled with the default loading steps in Chapter 3, the source file-name may be used after DBG. Otherwise the runfilename should be used.

4. Follow instructions in the the Source Level Debugger Guide. The following special definitions should be noted:

   - procedure-name and program-block-name mean the entire COBOL program named by program-id. Separately called programs would define other procedure-names or program-block-names.

   - labels mean paragraph-names and section-names. For the Debugger, these names must be preceded by a dollar sign. Thus, 040-EDIT would be entered as $040-EDIT.

   - Array elements are referred to by the array-name followed by the element number within parentheses. Multiple subscripts are separated by commas. Thus, element 3 of the array TABLE1 is referred to as TABLE1(3). Element 3 of the second dimension of TABLE2 would be TABLE2(2,3).

   - Elements with the same names must be referred to as:

     element-name OF group-name.

   - There is no way to set a breakpoint on a paragraph-name that is not unique. Paragraph-names, therefore, may not be qualified by section-names in breakpoint identifiers. Instead, set the breakpoint on the first statement in the paragraph.

   - DBG will not accept numeric literals longer than 14 digits.

   - LET cannot be used with edited data types.

   - LET cannot be used with an index name or an index data item.

   - LINAGE-COUNTER cannot be displayed.

   - Evaluation of abbreviated conditional expressions is not supported.

   - Results of arithmetic operations on scaled binary values are incorrect if decimal points are not aligned.

- Switches cannot be displayed.
- Assignments are not right-justified to justified data items.
- Any source line with "D" in column 7 will be executed.

5. To leave the Debugger, enter q.

EXAMPLES

This example uses the Debugger to examine data elements of the program OLDCASH at the end of Chapter 8. It uses three Debugger commands: BRK to set a breakpoint, the colon (:) to display data values, and LET to modify data values. Many other Debugger commands are discussed in the Source Level Debugger Guide.

The Debugger allows the program to present the usual requests for file assignments. If the program opens files, only C (continue) should be used after a breakpoint. RESTART may cause the program to attempt to open files that have not been closed.

```
OK, DBG OLDCASH

**Dbg**   revision x.x  (Date)

> BRK $040-EDIT
> RESTART
trace:  010-GET-JOBINFO
ENTER MONTH(ALPHA)
JUNE, 1982
ENTER JOB CODE
25
trace:  020-NEW-DETAIL-PAGE
trace:  150-NEW-PAGE
trace:  150-NEW-PAGE-EXIT
trace:  030-PROCESS-DETAIL
trace:  035-READ-AND-PRINT
**** breakpointed at DISBURSE\219 ($040-EDIT)
> : ENTRY-DETAIL
ENTRY-DETAIL.ENTRY-CHECK-NO = '408'
ENTRY-DETAIL.ENTRY-MONTH.ENTRY-MM = 8
ENTRY-DETAIL.ENTRY-MONTH.ENTRY-DD = 1
ENTRY-DETAIL.ENTRY-MONTH.ENTRY-YY = 78
ENTRY-DETAIL.FILLER = '   '
ENTRY-DETAIL.ENTRY-VENDOR = 'ASHTABULA HDWE
ENTRY-DETAIL.ENTRY-ACCT-NO = 430
ENTRY-DETAIL.ENTRY-AMOUNT = 354.76
> : TOTAL1
TOTAL1 = 0.00
> LET TOTAL1 = 111.11
> C
```

```
trace:  040-EDIT
trace:  050-DEPT-TOTALS
trace:  035-READ-AND-PRINT

**** breakpointed at DISBURSE\219 ($040-EDIT)
> : TOTAL1
TOTAL1 = 111.11
> Q
OK,
```

The next example illustrates the use of EDITOR commands within the Debugger environment. The command SRC NAME elicits the program-name of the program being debugged. The command SRC L PROC causes the editor to locate the first instance of PROCEDURE. The command SRC P15 causes the EDITOR to display 15 lines of source code.

```
OK, DBG OLDCASH

**Dbg**   revision x.x  (Date)

> SRC NAME
Source file is "<TPUBS>ANNE.K>NEWCBL>OLDCASH.CBL", based on evaluation
environment.
> SRC L PROC
  152:     PROCEDURE DIVISION.
> SRC P15
  152:     PROCEDURE DIVISION.
  153:    *
  154:     DECLARATIVES.
  155:     INPUT-ERROR SECTION. USE AFTER ERROR PROCEDURE ON DISK-FILE.
  156:     FIRST-PARAGRAPH.
  157:         DISPLAY  '**** I-O ERROR ON ENTRY: ***'.
  158:         DISPLAY  ENTRY-DETAIL, CLOSE DISK-FILE, PRINT-FILE.
  159:         STOP RUN.
  160:     TAPE-ERROR SECTION. USE AFTER ERROR PROCEDURE ON TAPE-FILE.
  161:     SECOND-PARAGRAPH.
  162:         DISPLAY '**** I-O ERROR ON TAPE OUTPUT ***'.
  163:         CLOSE DISK-FILE, PRINT-FILE.  STOP RUN.
  164:     END DECLARATIVES.
  165:    *
  166:     001-BEGIN.
> Q
OK,
```

E

Creating Indexed and Relative Files: The MIDASPLUS Interface

This appendix provides enough information for a user to create indexed and relative files with the minimum options needed to do indexed and relative I-O from a COBOL program. To meet this purpose, the appendix contains elementary instructions for MIDASPLUS (or the older MIDAS) files, followed by examples of the dialogs and data files needed as preparation to run the sample programs at the end of Chapters 12 and 13 and the update PTU98. For more information, see the MIDAS User's Guide.

> Caution
>
> Do not use COBOL 74 with an outdated revision of MIDAS or MIDASPLUS.

DEFINITIONS

The following terms are used in special ways in this appendix.

- Data file -- a nonindexed, nonrelative file of data records.
- Template or template file -- a file organized as an array with the array slots accessible only by indexes. The file may be empty.
- Indexed file -- a template file organized by indexes.

- Relative file -- a template file organized by indexes that correspond to record numbers.
- Direct access file -- same as a relative file.
- Input file -- a data file used to fill a template.
- Output file -- the template file.
- Create a file -- create an empty template file.
- Fill a file -- fill a template with data records.
- Build a file -- same as fill a file.

As indicated by this list of terms, preparation of a MIDASPLUS file usually involves two steps. To create an indexed or relative file, a user must first run an interactive program called CREATK to create a corresponding MIDASPLUS template for the file. To put records into that file, use either the KBUILD program or a COBOL program.

CREATK

CREATK has two dialogs:

- The "minimum options" dialog, which lets MIDAS supply default values for most of the structural parameters needed to build the template
- The "full options" dialog, which lets the user provide these parameter values

This appendix discusses only the minimum options dialog, which is sufficient for setting up either an indexed file or a direct access file. For the full options dialog, see the MIDAS User's Guide. Full options include:

```
A[DD]        ADD AN INDEX
D[ATA]       CHANGE DATA RECORD SIZE
E[XTEND]     CHANGE SEGMENT AND SEGMENT DIRECTORY LENGTH
F[ILE]       OPEN A NEW FILE
H[ELP]       PRINT THIS SUMMARY
M[ODIFY]     MODIFY AN EXISTING SUBFILE
P[RINT]      PRINT DESCRIPTOR INFORMATION
Q[UIT]       EXIT CREATK
(CR)         IMPLIED QUIT
S[IZE]       DETERMINE THE SIZE OF A FILE
U[SAGE]      DISPLAY CURRENT INDEX USAGE
V[ERSION]    MIDAS DEFAULTS FOR THIS FILE
```

KBUILD

There are at least three ways to build a file. One of these ways, the KBUILD utility, is documented here. The other methods are:

- Application programs
- Interactive entry
- Offline routines

They are covered as indicated in Table 3-1 of the MIDAS User's Guide. A quick comparison of these methods is also included there to help you decide which method is best suited to your needs.

KBUILD's Functions

KBUILD's range of functions is summarized below:

- Adding data to a new (empty) MIDASPLUS file template and building (adding entries to) the necessary index subfiles from sorted or unsorted input data.
- Adding new data and index entries to a MIDASPLUS file that already contains data entries.
- Adding entries from an external data file to a newly created secondary index subfile that has just been added to an existing (and previously filled) MIDASPLUS file.
- Converting a field from existing MIDASPLUS data subfile records to a secondary key field and adding these entries to a new or already existing secondary index subfile.

CREATK FOR INDEXED FILES

The CREATK Dialog

The following list of prompts from CREATK shows how to build an indexed file template with minimum options.

Prompt	Response	Remarks
OK, [CREATK rev x.x]	CREATK	
MINIMUM OPTIONS?	YES	Simplest options path.

Prompt	Response	Remarks
FILE NAME?	pathname	Enter pathname of indexed file to be used by the COBOL program. Pathnames are explained in the Prime User's Guide.
NEW FILE?	YES	For a new template.
DIRECT ACCESS?	NO	Create an indexed file instead.
DATA SUBFILE QUESTIONS		
PRIMARY KEY TYPE:	A,B,D,I,L,or S	A specifies an ASCII key. B specifies a bit-string key. D specified a double-precision floating-point key. I specifies a 16-bit or short integer key. L specifies a 32-bit or integer key. S specifies a single-precision floating-point key.
PRIMARY KEY SIZE = :	{ B / W } number	B number defines the number of bytes for an ASCII key or bits for a bit-string key. W number defines the number of halfwords for either kind of key. For example, if there are 2 characters in the key, number should be 16 bits, 2 bytes, or 1 halfword, depending on the key type. See the MIDAS User's Guide, Chapter 7, for the maximum key sizes. For D, four halfwords are required; for I, one halfword; for L and S, two halfwords.

Prompt	Response	Remarks
DATA SIZE = :	number	number is the number of halfwords for a data record, or the record length in characters, divided by 2 and rounded. It includes the key field for indexed files.

SECONDARY INDEX

Note

The following dialog is repeated for each alternate record key. The order in which alternate keys are specified to CREATK must be the same order as that listed for the ALTERNATE KEY clauses of the SELECT statement of the COBOL programs.

INDEX NO.?	number or CR	number is the number of the alternate record key or index. Carriage return (CR) will exit from CREATK, specifying no alternate indexes.
DUPLICATE KEYS PERMITTED?	YES or NO	YES allows the data in this key field to be duplicated. NO indicates that if the data in the key field is duplicated the file will not be updated and the INVALID KEY clause or the DECLARATIVES section will be activated.
KEY TYPE:	A, B, D, I, L, or S	Same as for Primary Key Type above.
KEY SIZE = :	{ B / W } number	Enter the size of the alternate key, determined in the same way as the primary key size above.
SECONDARY DATA SIZE = :	0	No data may be entered for secondary keys. The response must be 0, which will return the user to the prompt INDEX NO.? above.

Example

The following is a CREATK sequence for KDISBURS, the file used as MASTER-FILE by the sample program at the end of Chapter 12. KDISBURS has this record description:

```
01 MASTER-RECORD.
   05 ACCT-MS          PIC X(3).
   05 DATE-MS          PIC 9(6).
   05 FILLER           PIC X(3).
   05 VENDOR-MS        PIC X(20).
   05 CHECK-MS         PIC X(3).
   05 AMT-MS           PIC 9(7).
```

The record is 42 characters (21 halfwords) long. The primary key is ACCT-MS (three bytes). The dialog also creates two alternate keys, DATE-MS and CHECK-MS, for use with other programs. The alternate keys are six bytes and three bytes long.

```
OK, CREATK
[CREATK rev x.x]

MINIMUM OPTIONS? YES

FILE NAME? KDISBURS
NEW FILE? YES
DIRECT ACCESS? NO

DATA SUBFILE QUESTIONS

PRIMARY KEY TYPE: A
PRIMARY KEY SIZE = : B 3
DATA SIZE = : 21

SECONDARY INDEX

INDEX NO.? 1

DUPLICATE KEYS PERMITTED? YES

KEY TYPE: A
KEY SIZE = : B 6
SECONDARY DATA SIZE = : CR

INDEX NO.? 2

DUPLICATE KEYS PERMITTED? YES
```

```
KEY TYPE: A
KEY SIZE = : B 3
SECONDARY DATA SIZE = : CR

INDEX NO.? 0

SETTING FILE LOCK TO N READERS AND N WRITERS
OK,
```

KBUILD FOR INDEXED FILES

The KBUILD Dialog

The following dialog assumes that either every field in the input data file should be written to the output file, or that any unwanted fields are at the end of the record, so that they are not included in the record length in the fourth response below.

This dialog adds data to an empty template. To add or change an existing file, see other KBUILD options in the MIDAS User's Guide.

Prompt	Response	Remarks
OK,	KBUILD	
[KBUILD rev.x.x]		
SECONDARIES ONLY?	NO	To add data by primary or relative key.
ENTER INPUT FILENAME:	pathname	Enter name of nonindexed file containing data records.
ENTER INPUT RECORD LENGTH (WORDS):	number	Enter same number used for DATA SIZE with CREATK for the template file. This may be longer than the data record.
INPUT FILE TYPE:	COBOL	Specify a COBOL file.
ENTER NUMBER OF INPUT FILES:	number	If more than one data file is to be used, filenames must follow special conventions. See the MIDAS User's Guide.
ENTER OUTPUT FILENAME:	pathname	Enter pathname of template file to be used in the file assignment for the indexed file -- the template filename created with CREATK.
ENTER STARTING CHARACTER POSITION, PRIMARY KEY:	1	For COBOL, the primary key must start in position 1.

Note

The following two questions are repeated for each secondary or alternate key.

Prompt	Response	Remarks
SECONDARY NUMBER:	number or CR	Enter number of secondary keys (no more than in CREATK for the same file). Enter a carriage return to go on to the next part.
ENTER STARTING CHARACTER POSITION:	number	Enter character position of the start of that key within the record.

Note

The following dialog is used only once.

IS FILE SORTED?	YES or NO	Special rules apply to sorted files -- see the MIDAS User's Guide.
ENTER LOG/ERROR FILE:	pathname or CR	If CR is entered, no file is made and errors are only displayed on the screen.
ENTER MILESTONE COUNT:	number or CR	If a number is entered, KBUILD will display and log information for every nth record indexed. The 0 or CR causes first and last record only to be documented.

Example

The following is a KBUILD sequence for KDISBURS, the file created in the CREATK example above. The input data file, DISBURSE, is listed at the end of Chapter 12. The record is 21 halfwords long. The primary key must start in position 1. The secondary keys are DATE-MS and CHECK-MS, starting in positions 4 and 33, respectively.

```
OK, kbuild
[KBUILD rev x.x]
```

```
SECONDARIES ONLY? no
ENTER INPUT FILENAME: anne>disburse
ENTER INPUT RECORD LENGTH (WORDS): 21
INPUT FILE TYPE: text
ENTER NUMBER OF INPUT FILES: 1
ENTER OUTPUT FILENAME: anne.f>kdisburs
ENTER STARTING CHARACTER POSITION, PRIMARY KEY: 1
SECONDARY KEY NUMBER: 1
ENTER STARTING CHARACTER POSITION: 4
SECONDARY KEY NUMBER: 2
ENTER STARTING CHARACTER POSITION: 33
SECONDARY KEY NUMBER:
IS FILE SORTED? no
ENTER LOG/ERROR FILE NAME: klog
ENTER MILESTONE COUNT: 10

BUILDING: DATA
DEFERRING: 0, 1

PROCESSING FROM:  anne>disburse
 COUNT      DATE      TIME     CPU MIN   DISK MIN   TOTAL TM     DIFF
     0  06-18-82  09:01:01       0.000      0.000      0.000     0.000
    10  06-18-82  09:01:01       0.002      0.000      0.002     0.002
FIRST BUILD/DEFER PASS COMPLETE.
    10  06-18-82  09:01:01       0.003      0.000      0.003     0.000

SORTING INDEX 0
COUNT      DATE      TIME     CPU MIN   DISK MIN   TOTAL TM     DIFF
     0  06-18-82  09:01:01       0.000      0.000      0.000     0.000
SORT COMPLETE
    10  06-18-82  09:01:02       0.004      0.001      0.005     0.005

BUILDING INDEX 0
 COUNT      DATE      TIME     CPU MIN   DISK MIN   TOTAL TM     DIFF
     0  06-18-82  09:01:02       0.000      0.000      0.000     0.000
    10  06-18-82  09:01:02       0.001      0.000      0.001     0.001
INDEX 0 BUILT
    10  06-18-82  09:01:02       0.001      0.000      0.001     0.001

SORTING INDEX  1
 COUNT      DATE      TIME     CPU MIN   DISK MIN   TOTAL TM     DIFF
     0  06-18-82  09:01:03       0.000      0.000      0.000     0.000
SORT COMPLETE
    10  06-18-82  09:01:03       0.004      0.000      0.004     0.004

BUILDING INDEX   1
 COUNT      DATE      TIME     CPU MIN   DISK MIN   TOTAL TM     DIFF
     0  06-18-82  09:01:03       0.000      0.000      0.000     0.000
    10  06-18-82  09:01:03       0.001      0.000      0.001     0.001
INDEX  1 BUILT
    10  06-18-82  09:01:04       0.002      0.000      0.002     0.001

KBUILD COMPLETE.
```

CREATK FOR RELATIVE FILES

MIDASPLUS treats the relative key as a primary key. However, the character spaces allotted for the relative key in the CREATK and KBUILD dialog should not be included in the COBOL record description -- the key value will automatically be moved to the COBOL item defined as RELATIVE KEY. The following requirements for MIDASPLUS direct-access files should be noted:

- The relative key created by CREATK must have a length that is a multiple of 8 bits, between 8 and 48. The length of the key determines the number of records that may be in the file.
- In keyboard entry, description of literals, and all other operations, the relative key value should be entered with enough zero-fill to make a six-digit number.
- The maximum number of records to be allocated for the file must be specified when you create the file. The maximum possible is 999,999 if the primary key size is specified as B 48.
- The size and types of keys in COBOL programs need not be the same as those in the corresponding MIDASPLUS files.

CREATK Dialog

The following list of prompts shows use of CREATK to create a relative file with minimum options.

Prompt	Response	Remarks
OK, [CREATK rev x.x]	CREATK	
MINIMUM OPTIONS?	YES	Simplest options path.
FILE NAME?	pathname	Pathname of relative template file to be used in file assignments.
NEW FILE?	YES	For a new template.
DIRECT ACCESS?	YES	For a relative file.

Prompt	Response	Remarks
DATA SUBFILE QUESTIONS		
PRIMARY KEY TYPE:	B	B is required for COBOL.
PRIMARY KEY SIZE = :	B n	The maximum key size for a relative file is 48 bits, six characters (bytes), or three 16-bit halfwords. Key may always be specified at maximum size.
DATA SIZE = :	number	Length of record in characters, divided by 2 and rounded.
NUMBER OF ENTRIES TO ALLOCATE?	number	Number is the number of records to allocate in the new MIDASPLUS file.
SECONDARY INDEX		
INDEX NO.?	CR	No secondary keys are allowed. This concludes template creation and returns to command level.

Example

The following is a CREATK dialog for KMONTH, the master file used in the sample program at the end of Chapter 13. This template file has a record description of PIC X(35), so the data size here is 18. There is one record allotted for each month, so 12 is the maximum number of entries expected.

```
OK, creatk
[CREATK rev x.x]

MINIMUM OPTIONS? yes

FILE NAME? kmonth
NEW FILE? yes
DIRECT ACCESS? yes

DATA SUBFILE QUESTIONS

PRIMARY KEY TYPE: b
PRIMARY KEY SIZE = : b 48
DATA SIZE = : 18
NUMBER OF ENTRIES TO ALLOCATE? 12
```

```
SECONDARY INDEX

INDEX NO.? CR

SETTING FILE LOCK TO N READERS AND N WRITERS
OK,
```

KBUILD FOR RELATIVE FILES

The MIDASPLUS system requires that the relative index be included in the data record, even though COBOL codes the relative key separately from the record. The index should therefore be put at the end of the record in the data file.

The KBUILD Dialog

Prompt	Response	Remarks
OK,	KBUILD	
SECONDARIES ONLY?	NO	
ENTER INPUT FILENAME:	pathname	Enter name of sequential data file.
ENTER INPUT RECORD LENGTH (WORDS):	number	Enter number of character positions divided by 2, rounded.
INPUT FILE TYPE:	text	All COBOL files are text files.
ENTER NUMBER OF INPUT FILES:	number	If more than one data file is to be used, filenames must follow special conventions. See the MIDAS User's Guide.
ENTER OUTPUT FILENAME:	pathname	Enter name to be used for new relative file in file assignments -- the template filename created with CREATK.

Prompt	Response	Remarks
THE OUTPUT FILE SELECTED IS A DIRECT ACCESS FILE. IS THE ENTRY NUMBER SPECIFIED IN EACH INPUT RECORD AN ASCII STRING OR A BINARY (REAL*4) STRING? (ENTER A OR B):	a	All COBOL files are ASCII files.
ENTER STARTING CHARACTER POSITION IN INPUT RECORD:	number	Enter position number where the record number begins.
ENTER ENDING CHARACTER POSITION IN INPUT RECORD:	number	Enter position where the record number ends.
ENTER STARTING CHARACTER POSITION, PRIMARY KEY	number	Enter same number as for STARTING CHARACTER POSITION for input record above.
SECONDARY KEY NUMBER:	0 or CR	No secondary keys are allowed.
IS FILE SORTED?	yes or no	Special rules apply to sorted files -- see the MIDAS User's Guide.
ENTER LOG/ERROR FILE NAME:	pathname or CR	If CR is entered, no file is made and errors are only displayed on the screen.
ENTER MILESTONE COUNT:	number	KBUILD will display information for every nth data record put into the template file.

.
.
.

OK,

Example

The following example uses the template file KMONTH previously created with CREATK to build a relative file from the sequential file MONTHS, which is listed at the end of Chapter 13. MONTHS has 35-character (18-halfword) records with the record number in positions 30-35.

```
OK, kbuild
[KBUILD rev x.x]

SECONDARIES ONLY? no
ENTER INPUT FILENAME: anne.f>months
ENTER INPUT RECORD LENGTH (WORDS): 18
INPUT FILE TYPE: text
ENTER NUMBER OF INPUT FILES: 1
ENTER OUTPUT FILENAME: anne.f>kmonth
THE OUTPUT FILE SELECTED IS A DIRECT ACCESS FILE.
IS THE ENTRY NUMBER SPECIFIED IN EACH INPUT RECORD
AN ASCII STRING OR A BINARY (REAL*4) STRING? (ENTER A OR B): a
ENTER STARTING CHARACTER POSITION IN INPUT RECORD: 30
ENTER ENDING CHARACTER POSITION IN INPUT RECORD: 35
ENTER STARTING CHARACTER POSITION, PRIMARY KEY: 30
SECONDARY KEY NUMBER:CR
IS FILE SORTED? no
ENTER LOG/ERROR FILE NAME: klog
ENTER MILESTONE COUNT: 12

BUILDING: DATA
DEFERRING: 0

PROCESSING FROM:  anne.f>months
 COUNT      DATE      TIME     CPU MIN   DISK MIN   TOTAL TM     DIFF
       0  04-10-81  13:31:42   0.000      0.000      0.000      0.000
FIRST BUILD/DEFER PASS COMPLETE.
       7  04-10-81  13:31:44   0.006      0.001      0.007      0.001

SORTING INDEX 0
 COUNT      DATE      TIME     CPU MIN   DISK MIN   TOTAL TM     DIFF
       0  04-10-81  13:31:44   0.000      0.000      0.000      0.000
SORT COMPLETE
       7  04-10-81  13:31:45   0.005      0.002      0.007      0.007

BUILDING INDEX 0
 COUNT      DATE      TIME     CPU MIN   DISK MIN   TOTAL TM     DIFF
       0  04-10-81  13:31:45   0.000      0.000      0.000      0.000
INDEX 0 BUILT
       7  04-10-81  13:31:46   0.004      0.000      0.004      0.001

KBUILD COMPLETE.

OK,
```

F

COBOL 74 Library Files

To utilize COBOL 74, you must have the following files available in the UFDs specified:

UFD	File-name	Function
CMDNC0	CBL	Shared COBOL compiler
CMDNC0	PL1G	Shared PL1G compiler
SYSOVL	CBLDATA	Diagnostic file
LIB	CBLLIB	Shared COBOL 74 library
	or NCBLLIB	
	PFTNLB	Shared FORTRAN library
	IFTNLB	Nonshared FORTRAN library
SYSTEM	CB2154-60	Shared library segments
	CB2161	Shared library segments

The COBOL 74 library (CBLLIB or NCBLLIB) contains the following subroutines:

C$xxx	Routines used by both old and new COBOL
C$ADAT	Returns current date in format YYMMDD
C$ADAY	Returns Julian date in format YYDDD
C$ATIM	Returns current time in format HHMMSSFF: H = Hour M = Minutes S = Seconds F = Hundredths of seconds
C$CA	Closes sequential file

CCI/CCR	Closes indexed/relative file
C$CS	Closes sequential file
CDI/CDR	Deletes record from an indexed/relative file
CER/CER$	Error processing
CIN/CIN1/N$IN	File assignment initialization
C$INSP	INSPECT statement
C$KE	Updates file status code on an error
COI/COR	Opens indexed/relative file
C$OS	Opens sequential file
C$PRTN	Used with a EXIT PROGRAM to return to caller
CRI/CRR	Reads indexed/relative file
C$RS	Reads sequential file
CSI/CSR	Starts indexed/relative file
C$STR1/C$STR2/C$STR3	STRING statement
CSW/CSW0	Senses switch setting
C$UN	Unlocks an indexed/relative entry
C$UNS1/C$UNS2	UNSTRING statement
CWI/CWR	Writes indexed/relative file
C$WS	Writes a sequential file
CXI/CXR	Rewrites indexed/relative file
C$XS	Rewrites a sequential file
I$xxx	Midas routines
K$xxx	Midas library
N$xxx	COBOL 74 routines
N$ACLT	Alphabetic class test
N$ANY2	Runtime interface between object program and system conversion
N$ATOA	Interfaces to symbolic debugger
N$CRYPT	Not used in Rev. 18.4
N$CVRT	Miscellaneous conversion routine
N$DR	Deletes record from an indexed/relative file
N$FCHK	Checks file type and actual file (consistency between file and its FCB)
N$FID	Checks for magtape (runtime processor for VALUE OF FILE-ID IS data-name)
N$IN	File assignment initialization
N$INSP	INSPECT statement processor
N$NCLT	Numeric class test
N$OUT	Writes output to terminal
NRI/NRR	Reads indexed/relative file
NSI/NSR	Starts indexed/relative file
N$SMUT	Sort/Merge utility
N$STR	String statement processor
N$TIN	Reads input from terminal
N$TOUT	Writes output to user terminal
N$UNS	Unstring statement processor
N$WR	Writes indexed/relative file
N$WS	Writes a sequential file
N$XBTD	Binary to decimal conversion routine
N$XDTB	Decimal to binary conversion routine
N$XMV	Numeric move statement
N$XR	Rewrites indexed/relative file
N$ZED	Alphanumeric edited move
P$xxx	PL1G library

G
The MAP Option

The following sample listing for a large program includes a map. The listing includes the names of all data-items, program-name, section and paragraph headings, with the following information:

LEVEL — The level number specified by the user in the DATA description:

0 is for program, section, and paragraph names.

1 is for data-names in the DATA division.

SIZE — The size in 16-bit halfwords, unless followed by c indicating characters.

LOC — The memory address in octal notation, unless the -HEXADDRESS option is used. It may be followed by a number and a letter. If there is no number, the data is allocated in the link frame. Otherwise, the data is in a common block of that number. The common block names are at the end of the map.

ATTRIBUTES The first line shows whether the data-name is COMP-1, COMP-2, COMP-3, INDEX, DISPLAY, BINARY, or US-BINARY. The US means unsigned. The BINARY attribute corresponds to COBOL 74 data types in the following way:

BINARY-1 16-bit COMP, PIC 9 through PIC 9(4)

BINARY-2 32-bit COMP, PIC 9(5) through PIC 9(9)

BINARY-4 64-bit COMP, PIC 9(10) through PIC 9(18)

For DISPLAY items, the listing shows whether they have a separate sign or overpunch. It also signals group items.

The second line shows the line where the item is declared. It uses an asterisk for a line where the value of the item is changed.

For items from a copy file, the line number is shown in the format:

n<m>

where n is the line of the COPY statement in the main program, and m is the line number in the copy file.

In addition, at the end of the listing, the amount of 16-bit halfwords of working storage is given. Working storage is divided into:

- LINK BASE - the link frame space.
- The names of common blocks used. These names are created by the compiler, and end with a number plus a dollar sign to avoid possible use in a program.

The program below uses two common blocks, QWIB1$ and QWIB2$,created by the compiler with a hashing formula based on the program-id.

EXAMPLE

```
Source File:  <MFDXXX>ANNE>SAMPLES>REL2.CBL
Compiled on:  WED, APR 06 1983 at 17:56  by: CBL rev 7  03/07/83.23:04
Options are:  LISTING OPTIMIZE U(PPER)CASE MAP

    1           IDENTIFICATION DIVISION.
    2           PROGRAM-ID. REL2HUGE.
    3           ENVIRONMENT DIVISION.
    4           CONFIGURATION SECTION.
    5           SOURCE-COMPUTER. PRIME.
    6           OBJECT-COMPUTER. PRIME.
    7           INPUT-OUTPUT SECTION.
    8           FILE-CONTROL.
    9               SELECT A-FILE ASSIGN TO MT9
   10                       ORGANIZATION IS SEQUENTIAL.
   11                SELECT B-FILE ASSIGN TO PFMS
   12               ORGANIZATION IS SEQUENTIAL.
   13               SELECT C-FILE ASSIGN TO PFMS
   14               ORGANIZATION IS SEQUENTIAL.
   15               SELECT D-FILE ASSIGN TO PFMS
   16               ORGANIZATION IS SEQUENTIAL.
   17               SELECT T-FILE ASSIGN TO PFMS
   18                 ORGANIZATION IS INDEXED
   19                  ACCESS IS DYNAMIC
   20                          RECORD KEY IS T-KEY0
   21                          ALTERNATE RECORD KEY IS T-KEY1
   22                          ALTERNATE RECORD KEY IS T-KEY2
   23                          ALTERNATE RECORD KEY IS T-KEY3
   24                          FILE STATUS IS T-FILE-STATUS.
   25                      SELECT MIDAS-S-FILE ASSIGN TO PFMS
   26                         ORGANIZATION IS INDEXED
   27                         ACCESS IS DYNAMIC
   28                         RECORD KEY IS S-KEY0
   29                         ALTERNATE RECORD KEY IS S-KEY1 WITH DUPLICATES
   30                         ALTERNATE RECORD KEY IS S-KEY2 WITH DUPLICATES
   31                         FILE STATUS IS FILE-STATUS.
   32                      SELECT REL-1 ASSIGN TO PFMS
   33                         ORGANIZATION RELATIVE
   34                         ACCESS RANDOM
   35                         RELATIVE KEY REL-KEY
   36                         FILE STATUS REL-1-STATUS.
   37          DATA DIVISION.
   38          FILE SECTION.
   39          FD A-FILE
   40              LABEL RECORDS STANDARD
   41              VALUE OF FILE-ID IS 'YIKES'.
   42          01 A-REC.
   43              03 ATAB OCCURS 1000.
   44                05 A-ENT        PIC X(32).
   45          FD B-FILE
   46              LABEL RECORDS STANDARD
   47              VALUE OF FILE-ID IS 'B-FILE'.
   48          01 B-REC.
```

```
49              03 ATAB OCCURS 100.
50                05 A-ENT        PIC X(314).
51          FD C-FILE
52              LABEL RECORDS STANDARD
53              VALUE OF FILE-ID IS 'C-FILE'.
54          01 C-REC.
55              03 ATAB OCCURS 1000.
56                05 A-ENT        PIC X(32).
57          FD D-FILE
58              LABEL RECORDS STANDARD
59              VALUE OF FILE-ID IS 'D-FILE'.
60          01 D-REC.
61              03 ATAB OCCURS 1000.
62                05 A-ENT        PIC X(32).
63          FD T-FILE
64              LABEL RECORDS ARE STANDARD
65              VALUE OF FILE-ID IS 'TF-FILE1'.
66          01 TREC.
67              03 T-KEY0         PIC 9(4).
68              03 FILLER         PIC X.
69              03 T-KEY1         PIC 9(4).
70              03 T-KEY2         PIC 9(6).
71              03 T-KEY3         PIC 9(2).
72              03 T-DATA         PIC X(33).
73          FD MIDAS-S-FILE
74              LABEL RECORDS ARE STANDARD
75              VALUE OF FILE-ID IS SHORTREC-TREE.
76          01 SHORT-REC.
77              03 S-KEY0         PIC 9(5).
78              03 S-KEY1.
79                 05 S-KEY-1-X   PIC X.
80                 05 S-KEY1-9    PIC 9(6).
81              03 S-KEY2.
82                 05 S-KEY2-X    PIC X.
83                 05 K-KEY2-9    PIC 9(4).
84              03 S-DATA         PIC X(33).
85          FD REL-1 UNCOMPRESSED
86                 LABEL RECORD STANDARD
87                 VALUE OF FILE-ID IS REL-1-TREE.
88          01 RECORD-1.
89              02 PRIM-KEY       PIC 9(16).
90              02 ALT-KEY1       PIC 9(16).
91              02 ALT-KEY2       PIC 9(16).
92              02 FILLER         PIC X(12).
93          WORKING-STORAGE SECTION.
94          01 SHORTREC-TREE      PIC X(40) VALUE 'SHORTREC'.
95          01 RTREE.
96              03 FILLER         PIC X.
97              03 REL-1-TREE     PIC X(40) VALUE 'REL-1'.
98          01 REL-STATUS-STUFF.
99             03 FILLER          PIC X.
100            03 REL-1-STATUS    PIC X(2).
101            03 T-FILE-STATUS   PIC X(2).
102            03 FILE-STATUS     PIC X(2).
```

```
103             01 REL-KEY-STUFF.
104                03 FILLER          PIC X.
105                03 REL-KEY        PIC 9(6).
106             01 CPU-START          PIC X(4).
107             01 CPU-FIN            PIC X(4).
108             01 DISK-START         PIC X(4).
109             01 DISK-FIN           PIC X(4).
110             01 LOOP-COUNT          PIC S999 COMP VALUE 100.
111             PROCEDURE DIVISION.
112             DECLARATIVES.
113             DECLARE-1-SECTION SECTION.
114                 USE AFTER ERROR PROCEDURE ON REL-1.
115             DECLARE-1-P1.
116                 DISPLAY 'IO ERROR ON REL-1'.
117                 DISPLAY 'STATUS = '  REL-1-STATUS.
118                 DISPLAY 'REL-KEY = ' REL-KEY.
119                 CLOSE REL-1.
120                 GO TO ALL-DONE.
121             END DECLARATIVES.
122             START-SECTION SECTION.
123             P1.
124                 DISPLAY 'ENTER RELATIVE FILE PATHNAME'.
125                 ACCEPT REL-1-TREE.
126                 DISPLAY 'ENTER ISAM FILE NAME'.
127                 ACCEPT SHORTREC-TREE.
128                 OPEN INPUT REL-1.
129                 MOVE 0 TO PRIM-KEY.
130                 MOVE 0 TO REL-KEY.
131                 MOVE 0 TO ALT-KEY1.
132                 MOVE 0 TO ALT-KEY2.
133                 DISPLAY 'READ RELATIVE FILE TEST'.
134                 PERFORM READ-1 LOOP-COUNT TIMES.
135                 GO TO CLOSE-FILES.
136             READ-1.
137                 DISPLAY 'ENTER KEY AS 9(6) ITEM'.
138                 ACCEPT REL-KEY.
139                 READ REL-1 RECORD.
140                 DISPLAY RECORD-1.
141             CLOSE-FILES.
142                 CLOSE REL-1.
143             ALL-DONE.
144                 EXIT.
145             A100-MAIN.
146                 DISPLAY 'READ ISAM FILE TEST. (USING BYTE-ALIGNED KEY)'.
147                 DISPLAY 'ENTER Q$ TO QUIT'.
148                 OPEN I-O MIDAS-S-FILE.
149             A100-KEEP-GOING.
150                 DISPLAY 'ENTER KEY AS X(7) ITEM'.
151                 ACCEPT S-KEY1.
152                 IF S-KEY1 = 'Q$' THEN GO TO A999-END.
153                 READ MIDAS-S-FILE KEY IS S-KEY1
154                      INVALID KEY
155                      DISPLAY FILE-STATUS.
156                 EXHIBIT SHORT-REC.
```

```
157              GO TO A100-KEEP-GOING.
158          A999-END.
159              CLOSE MIDAS-S-FILE.
160              DISPLAY FILE-STATUS.
161          A200-MAIN.
162              DISPLAY 'READ ISAM FILE TEST (WORD ALIGNED KEY)'.
163              DISPLAY 'ENTER Q$ TO QUIT'.
164              OPEN I-O MIDAS-S-FILE.
165          A200-KEEP-GOING.
166              DISPLAY 'ENTER KEY AS X(5) ITEM'.
167              ACCEPT S-KEY2.
168              IF S-KEY2 = 'Q$' THEN GO TO A9999-END.
169              READ MIDAS-S-FILE KEY IS S-KEY2
170                 INVALID KEY
171                 DISPLAY FILE-STATUS.
172              EXHIBIT SHORT-REC.
173              GO TO A200-KEEP-GOING.
174          A9999-END.
175              CLOSE MIDAS-S-FILE.
176              DISPLAY FILE-STATUS.
177              STOP RUN.
```

DATA NAMES DECLARED IN REL2HUGE

NAME	LEVEL	SIZE	LOC (OCTAL)	ATTRIBUTES
A-FILE	FD	73	004406	FD-FILE DECLARED ON LINE 9
B-FILE	FD	84	103120	FD-FILE DECLARED ON LINE 11
C-FILE	FD	84	103244	FD-FILE DECLARED ON LINE 13
D-FILE	FD	84	103370	FD-FILE DECLARED ON LINE 15
T-FILE	FD	77	103514	FD-FILE DECLARED ON LINE 17
MIDAS-S-FILE	FD	73	103714	FD-FILE DECLARED ON LINE 25
REL-1	FD	74	104110	FD-FILE DECLARED ON LINE 32
A-REC	1	32000C	004517	ALPHANUMERIC GROUP ITEM REDEFINING ITEM DECLARED ON LINE 42
ATAB	3	32000C	004517	ALPHANUMERIC GROUP ITEM OCCURRING ITEM DECLARED ON LINE 43
A-ENT	5	32C	004517	ALPHANUMERIC DECLARED ON LINE 44
B-REC	1	31400C	000000	ALPHANUMERIC GROUP ITEM REDEFINING ITEM DECLARED ON LINE 48

```
ATAB            3     31400C C1+000431     ALPHANUMERIC GROUP ITEM
                                           OCCURRING ITEM
                                           DECLARED ON LINE 49
A-ENT           5     314C   C1+000431     ALPHANUMERIC
                                           DECLARED ON LINE 50
C-REC           1     32000C 075251        ALPHANUMERIC GROUP ITEM
                                           REDEFINING ITEM
                                           DECLARED ON LINE 54
ATAB            3     32000C C1+000431     ALPHANUMERIC GROUP ITEM
                                           OCCURRING ITEM
                                           DECLARED ON LINE 55
A-ENT           5     32C    C1+000431     ALPHANUMERIC
                                           DECLARED ON LINE 56
D-REC           1     32000C 000000        ALPHANUMERIC GROUP ITEM
                                           REDEFINING ITEM
                                           DECLARED ON LINE 60
ATAB            3     32000C C2+000433     ALPHANUMERIC GROUP ITEM
                                           OCCURRING ITEM
                                           DECLARED ON LINE 61
A-ENT           5     32C    C2+000433     ALPHANUMERIC
                                           DECLARED ON LINE 62
TREC            1     50C    103631        ALPHANUMERIC GROUP ITEM
                                           REDEFINING ITEM
                                           DECLARED ON LINE 66
T-KEY0          3     4C     103631        US-DISPLAY(4,0)
                                           DECLARED ON LINE 67
FILLER          3     1C     103633        ALPHANUMERIC
                                           DECLARED ON LINE 68
T-KEY1          3     4C     103633+1C     US-DISPLAY(4,0)
                                           DECLARED ON LINE 69
T-KEY2          3     6C     103635+1C     US-DISPLAY(6,0)
                                           DECLARED ON LINE 70
T-KEY3          3     2C     103640+1C     US-DISPLAY(2,0)
                                           DECLARED ON LINE 71
T-DATA          3     33C    103641+1C     ALPHANUMERIC
                                           DECLARED ON LINE 72
SHORT-REC       1     50C    104025        ALPHANUMERIC GROUP ITEM
                                           REDEFINING ITEM
                                           DECLARED ON LINE 76
S-KEY0          3     5C     104025        US-DISPLAY(5,0)
                                           DECLARED ON LINE 77
S-KEY1          3     7C     104027+1C     ALPHANUMERIC GROUP ITEM
                                           DECLARED ON LINE 78
S-KEY-1-X       5     1C     104027+1C     ALPHANUMERIC
                                           DECLARED ON LINE 79
S-KEY1-9        5     6C     104030        US-DISPLAY(6,0)
                                           DECLARED ON LINE 80
S-KEY2          3     5C     104033        ALPHANUMERIC GROUP ITEM
                                           DECLARED ON LINE 81
S-KEY2-X        5     1C     104033        ALPHANUMERIC
                                           DECLARED ON LINE 82
K-KEY2-9        5     4C     104033+1C     US-DISPLAY(4,0)
                                           DECLARED ON LINE 83
S-DATA          3     33C    104035+1C     ALPHANUMERIC
```

				DECLARED ON LINE 84
RECORD-1	1	60C	104222	ALPHANUMERIC GROUP ITEM REDEFINING ITEM DECLARED ON LINE 88
PRIM-KEY	2	16C	104222	US-DISPLAY(16,0) DECLARED ON LINE 89
ALT-KEY1	2	16C	104232	US-DISPLAY(16,0) DECLARED ON LINE 90
ALT-KEY2	2	16C	104242	US-DISPLAY(16,0) DECLARED ON LINE 91
FILLER	2	12C	104252	ALPHANUMERIC DECLARED ON LINE 92
SHORTREC-TREE	1	40C	104317	ALPHANUMERIC DECLARED ON LINE 94
RTREE	1	41C	104343	ALPHANUMERIC GROUP ITEM DECLARED ON LINE 95
FILLER	3	1C	104343	ALPHANUMERIC DECLARED ON LINE 96
REL-1-TREE	3	40C	104343+1C	ALPHANUMERIC DECLARED ON LINE 97
REL-STATUS-STUFF	1	7C	104370	ALPHANUMERIC GROUP ITEM DECLARED ON LINE 98
FILLER	3	1C	104370	ALPHANUMERIC DECLARED ON LINE 99
REL-1-STATUS	3	2C	104370+1C	ALPHANUMERIC DECLARED ON LINE 100
T-FILE-STATUS	3	2C	104371+1C	ALPHANUMERIC DECLARED ON LINE 101
FILE-STATUS	3	2C	104372+1C	ALPHANUMERIC DECLARED ON LINE 102
REL-KEY-STUFF	1	7C	104374	ALPHANUMERIC GROUP ITEM DECLARED ON LINE 103
FILLER	3	1C	104374	ALPHANUMERIC DECLARED ON LINE 104
REL-KEY	3	6C	104374+1C	US-DISPLAY(6,0) DECLARED ON LINE 105
CPU-START	1	4C	104400	ALPHANUMERIC DECLARED ON LINE 106
CPU-FIN	1	4C	104402	ALPHANUMERIC DECLARED ON LINE 107
DISK-START	1	4C	104404	ALPHANUMERIC DECLARED ON LINE 108
DISK-FIN	1	4C	104406	ALPHANUMERIC DECLARED ON LINE 109
LOOP-COUNT	1	1	104410	BINARY-1(10,0) DECLARED ON LINE 110

PROCEDURE NAMES DEFINED IN REL2HUGE

NAME	ATTRIBUTES
DECLARE-1-SECTION	SECTION END OF PERFORM RANGE

DECLARE-1-P1	DECLARED ON LINE 113 PARAGRAPH
START-SECTION	DECLARED ON LINE 115 SECTION
P1	DECLARED ON LINE 122 PARAGRAPH
READ-1	DECLARED ON LINE 123 PARAGRAPH END OF PERFORM RANGE
CLOSE-FILES	DECLARED ON LINE 136 PARAGRAPH
ALL-DONE	DECLARED ON LINE 141 PARAGRAPH
A100-MAIN	DECLARED ON LINE 143 PARAGRAPH
A100-KEEP-GOING	DECLARED ON LINE 145 PARAGRAPH
A999-END	DECLARED ON LINE 149 PARAGRAPH
A200-MAIN	DECLARED ON LINE 158 PARAGRAPH
A200-KEEP-GOING	DECLARED ON LINE 161 PARAGRAPH
A9999-END	DECLARED ON LINE 165 PARAGRAPH
	DECLARED ON LINE 174

PROGRAMS CALLED FROM REL2HUGE

(NONE)

LINK BASE SIZE 34825 HALFWORDS

COMMON (EXTERNAL) AREAS

QWIB1$	63402	HALFWORDS IN AREA
QWIB2$	32001	HALFWORDS IN AREA

DIAGNOSTIC SUMMARY

H

The XREF Option

Listed below is a sample program followed by a cross-reference listing created with the XREFSORT compile option. The program includes two COPY statements, one in the WORKING-STORAGE section, and one in the PROCEDURE division.

The cross listing includes all features provided by the -MAP option. In addition, it provides a list of all lines where each data-item is referenced. XREF causes the names in the map to be listed in source program order. XREFSORT causes the names to be listed in alphanumeric order. It gives each data-name, followed by:

LEVEL — The level-number specified by the user in the DATA description.

SIZE — The size in 16-bit halfwords, unless followed by c indicating characters.

LOC — The memory address in octal notation, unless the -HEXADDRESS option is used, and may be followed by a two-character code. If there is no number, the data is allocated in the link frame. Any other number is the number of the common block at the end of the listing.

ATTRIBUTES The first line shows whether the data-name is COMP-1, COMP-2, COMP-3, INDEX, DISPLAY, BINARY, or US-BINARY. The US means unsigned. The BINARY attribute corresponds to COBOL 74 data types in the following way:

BINARY-1 16-bit COMP, PIC 9 through PIC 9(4)

BINARY-2 32-bit COMP, PIC 9(5) through PIC 9(9))

BINARY-4 64-bit COMP, PIC 9(10) through PIC 9(18)

For DISPLAY items, the listing shows whether they have a separate sign or overpunch. It also signals group items. The precision of the data items is also shown.

The second line shows the line where the item is declared and all lines where it is referred to. Line numbers prefixed with an asterisk indicate destructive references.

For items from a copy file, the line number is shown in the format:

n<m>

where n is the line of the COPY statement in the main program, and m is the line number in the copy file.

EXAMPLE

```
Source File:  <MFDXXX>ANNE>SAMPLES>XREF3.CBL
Compiled on:  WED, APR 06 1983 at 17:58  by: CBL rev 7  03/07/83.23:04
Options are:  LISTING OPTIMIZE XREF U(PPER)CASE MAP MAPSORT

    1           IDENTIFICATION DIVISION.
    2           PROGRAM-ID. XREF3.
    3          ******************************************
    4           ENVIRONMENT DIVISION.
    5           CONFIGURATION SECTION.
    6           SOURCE-COMPUTER. PRIME.
    7           OBJECT-COMPUTER. PRIME.
    8          ******************************************
    9           DATA DIVISION.
   10               COPY 'XREF3.LIB'.
<     1>        WORKING-STORAGE SECTION.
<     2>        01 LEADING-SEP      PIC S99 COMP VALUE 51.
<     3>        01 TRAILING-SEP     PIC S99 COMP VALUE 52.
<     4>        01 FIXED-DEC        PIC S99 COMP VALUE 4.
<     5>        01 LEADING-OVP      PIC S99 COMP VALUE 54.
<     6>        01 TRAILING-OVP     PIC S99 COMP VALUE 55.
<     7>        01 US-DISPLAY       PIC S99 COMP VALUE 63.
```

```
<    8>          01 INDEX-NAME       PIC S99 COMP VALUE 67.
<    9>          01 INDEX-ITEM       PIC S99 COMP VALUE 67.
<   10>          01 LS               PIC S9(7)V9(2) LEADING SEPARATE
<   11>                     VALUE -1234567.89.
<   12>          01 TS               PIC S9(7)V9(2) TRAILING SEPARATE
<   13>                     VALUE +9876543.21.
<   14>          01 C3               PIC S9(7)V9(2) COMP-3
<   15>                     VALUE +0.
<   16>         *
<   17>          01 LOP              PIC S9(7)V9(2) LEADING
<   18>                     VALUE -7654321.98.
<   19>          01 TROP             PIC S9(7)V9(2) TRAILING
<   20>                     VALUE +23.45.
   11            01 USD                    PIC 9(7)V9(2)
   12                                              VALUE 5544773.32.
   13            01 IND-DATA-ITEM USAGE IS INDEX.
   14            01 TAB.
   15               05 T-ELEMENT PIC 99 COMP OCCURS 10 INDEXED BY
   16                                    INDX.
   17            01 FB1          PIC S99 COMP VALUE 32765.
   18            01 FB2          PIC S99999 COMP VALUE 1234567.
   19            01 FB4        PIC S99999999999 COMP VALUE 12345678901.
   20            01 USFB1         PIC 99  COMP VALUE 3000.
   21            01 USFB2            PIC 999999 COMP VALUE 223344.
   22            01 USFB4         PIC 999999999999999 COMP
   23                                       VALUE 123451234512345.
   24            01 CMP1          COMP-1 VALUE 2345.67E+4.
   25            01 CMP           COMP-2 VALUE -764321.98E+14.
   26            01 SB            PIC S9(2)V9(2) COMP VALUE 76.54.
   27            01 SB2           PIC S9(3)V9(3) COMP VALUE 987.654.
   28            01 SB4  PIC S9(12)V9(4) COMP VALUE -555554444433.1234.
   29            01 P             PIC S99 COMP VALUE ZERO.
   30            01 Q             PIC S99 COMP VALUE ZERO.
   31            01 TY            PIC S99 COMP VALUE ZERO.
   32            01 PVALUE        PIC S9 COMP VALUE 9.
   33            01 QVALUE        PIC S9 COMP VALUE 2.
   34            01 CVAR.
                    05 CSIZE     PIC S9 COMP VALUE 0.
                    05 CSTRING   PIC X(30) VALUE SPACES.
   37            01 FAKE1         PIC S9(5) VALUE 000000015.
   38            01 FAKE2         PIC S9(4)V9(2) VALUE -0000001234.56.
   39            01 FAKE3         PIC S9(4)V9(2) VALUE +00000001234.56.
   40            01 LSA           PIC S9(7)V9(2) LEADING SEPARATE
   41                             VALUE -1234567.89.
   42            01 TSA           PIC S9(7)V9(2) TRAILING SEPARATE
   43                              VALUE +9876543.21.
   44            01 C3A                  PIC S9(7)V9(2) COMP-3
   45                 VALUE +0.
   46           *
   47            01 LOPA                 PIC S9(7)V9(2) LEADING
   48                            VALUE -7654321.98.
   49            01 TROPA                PIC S9(7)V9(2) TRAILING
   50                             VALUE +23.45.
   51            01 USDA                 PIC 9(7)V9(2)
```

```
   52                                VALUE 5544773.32.
   53            01 FB1A             PIC S99 COMP VALUE 32765.
   54            01 FB2A             PIC S99999 COMP VALUE 123567.
   55            01 FB4A       PIC S99999999999 COMP VALUE 12345678901.
   56            01 USFB1A     PIC 99  COMP VALUE 3000.
   57            01 USFB2A     PIC 999999 VALUE 223344.
   58            01 USFB4A     PIC 999999999999999 VALUE 123451234512345.
   59            01 CMP1A         COMP-1 VALUE 2345.67E+4.
   60            01 CMP2A         COMP-2 VALUE -764321.98E+14.
   61            01 SB1A           PIC S9(2)V9(2) COMP VALUE 76.54.
   62            01 SB2A           PIC S9(3)V9(3) COMP VALUE 987.654.
   63            01 SB4A           PIC S9(12)V9(4) COMP VALUE -5555544444
33.1234.
   64           ************************************************************
   65            PROCEDURE DIVISION.
   66            S1 SECTION.
   67            P1.
   68                MOVE LS TO LSA.
   69                MOVE LS TO TSA.
   70                COPY 'XREF2.LIB'.
<     1>            MOVE TS TO C3A.
<     2>            MOVE TS TO LOPA.
<     3>            MOVE TS TO TROP.
<     4>            MOVE TS TO USDA.
<     5>            MOVE TS TO FB1A.
<     6>            MOVE TS TO FB2A.
<     7>            MOVE TS TO FB4A.
<     8>            MOVE TS TO USFB1A.
<     9>            MOVE TS TO USFB2A.
<    10>            MOVE TS TO USFB4A.
<    11>            MOVE TS TO CMP1A.
<    12>            MOVE TS TO SB1A.
<    13>            MOVE SB2 TO SB2A.
<    14>            MOVE SB4 TO SB4A.
```

DATA NAMES DECLARED IN XREF3

NAME	LEVEL	SIZE	LOC (OCTAL)	ATTRIBUTES ("*" = DESTRUCTIVE REF)
C3	1	5C	004342	COMP-3(9,2) DECLARED ON LINE 10<14>
C3A	1	5C	004511	COMP-3(9,2) DECLARED ON LINE 44 REFERENCES: *70<1>
CMP	1	4	004426	COMP-2(47,0) DECLARED ON LINE 25
CMP1	1	2	004424	COMP-1(23,0) DECLARED ON LINE 24
CMP1A	1	2	004556	COMP-1(23,0) DECLARED ON LINE 59 REFERENCES: *70<11>
CMP2A	1	4	004560	COMP-2(47,0) DECLARED ON LINE 60
CSIZE	5	1	004446	BINARY-1(4,0) COMPILER-ALIGNED DECLARED ON LINE 35
CSTRING	5	30C	004447	ALPHANUMERIC DECLARED ON LINE 36
CVAR	1	32C	004446	ALPHANUMERIC GROUP ITEM DECLARED ON LINE 34
FAKE1	1	5C	004466	TRAILING OVP(5,0) DECLARED ON LINE 37
FAKE2	1	6C	004471	TRAILING OVP(6,2) DECLARED ON LINE 38
FAKE3	1	6C	004474	TRAILING OVP(6,2) DECLARED ON LINE 39
FB1	1	1	004406	BINARY-1(7,0) DECLARED ON LINE 17
FB1A	1	1	004533	BINARY-1(7,0) DECLARED ON LINE 53 REFERENCES: *70<5>
FB2	1	2	004407	BINARY-2(17,0) DECLARED ON LINE 18
FB2A	1	2	004534	BINARY-2(17,0) DECLARED ON LINE 54 REFERENCES: *70<6>
FB4	1	4	004411	BINARY-4(37,0) DECLARED ON LINE 19
FB4A	1	4	004536	BINARY-4(37,0) DECLARED ON LINE 55 REFERENCES: *70<7>
FIXED-DEC	1	1	004322	BINARY-1(7,0) DECLARED ON LINE 10<4>
IND-DATA-ITEM	1	4	004364	INDEX-DATA-NAME DECLARED ON LINE 13
INDEX-ITEM	1	1	004327	BINARY-1(7,0) DECLARED ON LINE 10<9>
INDEX-NAME	1	1	004326	BINARY-1(7,0)

```
                                                DECLARED ON LINE 10<8>
INDX                 4      004402              INDEX-NAME
                                                DECLARED ON LINE 16
LEADING-OVP     1    1      004323              BINARY-1(7,0)
                                                DECLARED ON LINE 10<5>
LEADING-SEP     1    1      004320              BINARY-1(7,0)
                                                DECLARED ON LINE 10<2>
LOP             1    9C     004345              LEADING OVP(9,2)
                                                DECLARED ON LINE 10<17>
LOPA            1    9C     004514              LEADING OVP(9,2)
                                                DECLARED ON LINE 47
                                                REFERENCES:  *70<2>
LS              1    10C    004330              LEADING SEP(9,2)
                                                DECLARED ON LINE 10<10>
                                                REFERENCES:  68 69
LSA             1    10C    004477              LEADING SEP(9,2)
                                                DECLARED ON LINE 40
                                                REFERENCES:  *68
P               1    1      004441              BINARY-1(7,0)
                                                DECLARED ON LINE 29
PVALUE          1    1      004444              BINARY-1(4,0)
                                                DECLARED ON LINE 32
Q               1    1      004442              BINARY-1(7,0)
                                                DECLARED ON LINE 30
QVALUE          1    1      004445              BINARY-1(4,0)
                                                DECLARED ON LINE 33
SB              1    1      004432              BINARY-1(14,7)
                                                DECLARED ON LINE 26
SB1A            1    1      004564              BINARY-1(14,7)
                                                DECLARED ON LINE 61
                                                REFERENCES:  *70<12>
SB2             1    2      004433              BINARY-2(20,10)
                                                DECLARED ON LINE 27
                                                REFERENCES:  70<13>
SB2A            1    2      004565              BINARY-2(20,10)
                                                DECLARED ON LINE 62
                                                REFERENCES:  *70<13>
SB4             1    4      004435              BINARY-4(54,14)
                                                DECLARED ON LINE 28
                                                REFERENCES:  70<14>
SB4A            1    4      004567              BINARY-4(54,14)
                                                DECLARED ON LINE 63
                                                REFERENCES:  *70<14>
T-ELEMENT       5    10     004370              US-BINARY-1(7,0) OCCURRING ITEM
                                                DECLARED ON LINE 15
TAB             1    10     004370              ALPHANUMERIC GROUP ITEM
                                                DECLARED ON LINE 14
TRAILING-OVP    1    1      004324              BINARY-1(7,0)
                                                DECLARED ON LINE 10<6>
TRAILING-SEP    1    1      004321              BINARY-1(7,0)
                                                DECLARED ON LINE 10<3>
TROP            1    9C     004352              TRAILING OVP(9,2)
                                                DECLARED ON LINE 10<19>
                                                REFERENCES:  *70<3>
```

```
TROPA           1     9C     004521     TRAILING OVP(9,2)
                                        DECLARED ON LINE 49
TS              1     10C    004335     TRAILING SEP(9,2)
                                        DECLARED ON LINE 10<12>
                                        REFERENCES: 70<1> 70<2> 70<3>
                                        70<4> 70<5> 70<6> 70<7> 70<8>
                                        70<9> 70<10> 70<11> 70<12>
TSA             1     10C    004504     TRAILING SEP(9,2)
                                        DECLARED ON LINE 42
                                        REFERENCES:  *69
TY              1     1      004443     BINARY-1(7,0)
                                        DECLARED ON LINE 31
US-DISPLAY      1     1      004325     BINARY-1(7,0)
                                        DECLARED ON LINE 10<7>
USD             1     9C     004357     US-DISPLAY(9,2)
                                        DECLARED ON LINE 11
USDA            1     9C     004526     US-DISPLAY(9,2)
                                        DECLARED ON LINE 51
                                        REFERENCES:  *70<4>
USFB1           1     1      004415     US-BINARY-1(7,0)
                                        DECLARED ON LINE 20
USFB1A          1     1      004542     US-BINARY-1(7,0)
                                        DECLARED ON LINE 56
                                        REFERENCES:  *70<8>
USFB2           1     2      004416     US-BINARY-2(20,0)
                                        DECLARED ON LINE 21
USFB2A          1     6C     004543     US-DISPLAY(6,0)
                                        DECLARED ON LINE 57
                                        REFERENCES:  *70<9>
USFB4           1     4      004420     US-BINARY-4(50,0)
                                        DECLARED ON LINE 22
USFB4A          1     15C    004546     US-DISPLAY(15,0)
                                        DECLARED ON LINE 58
                                        REFERENCES:  *70<10>

PROCEDURE NAMES DEFINED IN XREF3

NAME                                    ATTRIBUTES    ("*" = DESTRUCTIVE
REF)

P1                                      PARAGRAPH
                                        DECLARED ON LINE 67
S1                                      SECTION
                                        DECLARED ON LINE 66

PROGRAMS CALLED FROM XREF3

(NONE)


LINK BASE SIZE      2171    HALFWORDS
```

I

Prime Support of the ANSI Standard

Module	Features Available in Prime COBOL 74
Nucleus	Full level 2, with these exceptions: ALPHABET-NAME IS LITERAL is not supported. ENTER is checked for syntax only. LINAGE, LINAGE-COUNTER, PAGE-COUNTER, END-OF-PAGE and EOP are not supported. STRING and UNSTRING cannot have more than five sending or receiving fields. SAME AREA is treated as SAME RECORD AREA. MEMORY SIZE is not supported. The OPTIONAL attribute is syntax-checked only. COLLATING SEQUENCE recognizes only ASCII and EBCDIC.
Sequential I-O	Full level 2, with these exceptions: RERUN, MULTIPLE FILE, REWIND, REMOVAL, REEL, REVERSED, UNIT, and LOCK are not supported.
Relative I-O	Full level 2, with minor exceptions required for: MIDASPLUS interface. RERUN is checked for syntax only.

Indexed I-O	Full level 2, with minor exceptions required for:

MIDAS+ interface.
RERUN is checked for syntax only.

Sort-merge	Full level 2

SORT or MERGE with the USING option is not supported for tape files.

Library	Full level 2
Table Handling	Full level 2, with this exception:

Variable-length tables are implemented only for use with SEARCH.

Interprogram Communication	Full level 2, with these exceptions:

CALL data-name is not supported.
CANCEL is not supported.
ON OVERFLOW is checked for syntax only.

PRIME EXTENSIONS TO THE ANSI STANDARD

Prime COBOL 74 provides the following extensions to ANSI COBOL-74:

- Single quote or double quote allowed as the delimiter for literals
- Lowercase interpreted as uppercase except in nonnumeric literals and in alphabetic class tests
- Comment lines preceding the IDENTIFICATION DIVISION
- ID for IDENTIFICATION
- REMARKS
- Optional paragraphs in the IDENTIFICATION division appearing in any order
- Optional paragraphs in the I-O-CONTROL section appearing in any order
- SOURCE-COMPUTER and OBJECT-COMPUTER not required
- FDs and SDs in any order
- The LABEL RECORDS clause not required

- UNCOMPRESSED or COMPRESSED file attribute
- In records, level-numbers greater than 1 appearing in any order
- Subscripts that may themselves be subscripted
- In documentation, the term data-name referring to an item that may be indexed or subscripted
- COMPUTATIONAL-1, COMPUTATIONAL-2 (COMP-1, COMP-2) (floating-point formats)
- COMPUTATIONAL-3 (COMP-3) (packed decimal format)
- ALTERNATE RECORD KEY nonnumeric
- FILE STATUS as PIC 99 or PIC XX
- The RELATIVE KEY clause appearing anywhere in the file-control entry
- FILLER used at the group level
- COMP with a default PICTURE of S9999
- Subscripting and qualification of data-names listed as subscripts
- Arithmetic expressions as subscripts
- Eight levels of subscripting
- Paragraph-names and section-names not required in the PROCEDURE division
- CORRESPONDING option for IF, MULTIPLY, DIVIDE, COMPUTE
- The ROUNDED and SIZE ERROR options for numeric MOVEs
- Arithmetic expressions after GO TO DEPENDING, IF, MOVE, PERFORM VARYING, SET, in indexing, in subscripting, and in all arithmetic statements
- EXHIBIT NAMED
- NOTE
- READY TRACE and RESET TRACE
- SEEK (checked for syntax only)
- EJECT, SKIP1, SKIP2, and SKIP3
- GOBACK

- DISPLAY ... NO ADVANCING
- DISPLAY with multiple operands
- THEN and OTHERWISE after IF
- Literals as operands of INSPECT
- Contents of record area undisturbed after REWRITE or WRITE
- Arguments of any level number in a CALL statement
- Cross reference listing
- Map listing
- FIPS checking
- Subscript range checking
- Interface to the Prime Source Level Debugger
- PROCEDURE division references to PROGRAM-ID and DATE-COMPILED.
- A data-name that redefines another may itself be the object of a REDEFINES clause.
- Underscore allowed in data-names
- Right margin extended with -RMARGIN

J

Implementation-dependent Features of Prime COBOL 74 in Rev. 18

Maximum Sizes

Unpacked Decimal (DISPLAY) number	36 digits
Packed Decimal (COMP-3) number	36 digits
Binary (COMP) number	18 digits, default S9999
Floating-point-1 (COMP-1) value	10E38, minimum 10E-38
Mantissa	7 digits
Floating-point-2 (COMP-2) value	10E+9823, minimum 10E-9902
Mantissa	14 digits
Index value (max occurrence number)	64K(65,535) if referring to a two-character item 128K(131,071) if referring to a one-character item
Index size	64 bits
WORKING-STORAGE size	100 128K-byte segments
Program size (PROCEDURE division)	128K bytes (one segment)
Table (array) size	128K bytes (characters)
Length of data-names and other programmer-defined words	30 characters
File name as literal, including pathname	8 characters for -OLD option, 120 characters for normal I-O
Program-id	8 characters recognized
Size of an elementary item	64K bytes
Record size:	
in file	64K bytes
in working storage	128K bytes
Block size	32K records (if each record is only 16 bits)

Maximum Numbers

Characters in ACCEPT or DISPLAY	256
Number of subscripts (array dimensions)	8
Number of secondary keys	17
Number of files open at once	128 (0 and 127 are reserved for PRIMOS)
Number of files merged	2-11
Number of files sorted	1-20
Number of qualifiers:	
for paragraph-names	1
for data-names or condition-names	50
Operands of STRING and UNSTRING	5
Operands of USING	64

Other Information

High values	hex FF
Low values	hex 00

K

File Assignments with -OLD

INTERACTIVE ASSIGNMENTS

If the -OLD option is used for compilation, interactive file assignments must be made under the following conditions:

- No EXIT PROGRAM statement is included in the program.
- FD's are contained in the runfile.
- The VALUE OF FILE-ID clause for one or more FDs contains a literal rather than a data-name.

In this case, immediately following the execute command SEG runfilename or EXECUTE, a request is displayed for runtime file assignments:

```
ENTER FILE ASSIGNMENTS:
>
```

For files whose names within the program are incomplete, give the literal from the VALUE OF FILE-ID clause of a file description, followed by an equals sign, the name of the actual file to be associated with the program file-name, and a carriage return:

```
> name-in-literal = disk-or-tape-name    (CR)
```

The formats for disk and tape file-names are given below.

If file-names and pathnames in the FD entry are complete, simply enter a slash mark (/) to the request.

The system will display the prompt character > while waiting for more user input. The user should make one entry for each FD whose FILE-ID is to be reassigned. When no file assignments remain to be entered, use the slash mark to conclude the session. Execution of the application program will then begin, using the file assignments that were just entered. The existence and type of each file are checked when OPEN is executed for that file.

Default Assignments

If a VALUE OF FILE-ID is present and no interactive assignment is made, then the actual file-name is the name specified in VALUE OF FILE-ID. If the VALUE OF FILE-ID clause contains a literal, only the first eight characters are used. This means that unlabelled tape file assignments cannot be made within the literal. If the clause contains a data-name, then the file pathname should be assigned to that data-name by the COBOL program.

If no VALUE OF FILE-ID clause is present for a file, the compiler generates output files or searches for input files with the names F1, F2, F3, and so on, up to nine files. Different names are generated for print files as explained below.

> **Caution**
>
> This second method of default assignment is not recommended for new programs. It is intended only for compatibility with programs transported from other compilers.

Print files follow a different default naming convention. Files assigned to the printer are assigned by the COBOL runtime package to names consisting of the first four characters of the program-name plus a two-digit sequence number. The sequence number is 01 the first time the program is run. It is incremented by one each time the program is executed, if previous print files still exist. Thus when the program with program-id DISBURSE is run the first time, the print file is named DISB01 by default. The second run creates DISB02 if DISB01 still exists.

FILE-NAME FORMATS FOR DISK AND TAPE

This section gives the proper format for PRIMOS file-names, whether in a literal or a data-name after VALUE OF FILE-ID, or in the runtime file assignments.

Disk File-Name Format

A pathname for a disk file describes the location of the file in the directory hierarchy. It consists of an optional partition-name preceded by the symbol < , followed optionally by the UFD-name followed by any sub-UFDs, followed by the filename, all separated by the symbol >. If no UFD is specified, the current UFD is searched. UFDs and pathnames are explained in the Prime User's Guide.

Pathnames specified in file assignments should not contain spaces. If a space must be specified in order to include a password, enclose the entire pathname in quotes.

Examples of pathnames are:

```
<PART1>COBOL>FILE2
UFD1>FILE
UFD2>DATA>FILE1
'UFD1 PASSWORD>FILE'
```

The FILE-ID clause may be used in conjunction with the OWNER IS clause (Chapter 7). In this case, the FILE-ID must be followed by a literal. If both FILE-ID IS literal-2 and OWNER IS literal-1 are used, the pathname sought is literal-1>literal-2.

Tape Assignment Format

To specify the location of a tape file, you must know the drive, the name of the tape volume, and, for files used as input, the owner-id. The format for a tape file assignment is:

```
drivename, label-type, owner-id, volume-id
```

drivename	$MTx, where x is a drive number from 0 through 7 (0 through 7 if logical drives were assigned)
label-type	N: no label information S: standard labels
owner-id	A 14-character field. This is also called the tape file-id.
volume-id	A six-character field that is written in the label of the tape being created, or is checked if the tape is being read. This is also called the volume serial number (VSN).

Chapter 14 discusses tape files more thoroughly.

Assignment Examples for -OLD

With -OLD, files on disk or tape can be associated with FDs in the program either by runtime file assignments or by COBOL statements within the program:

- Suppose that in a COBOL program the following statements existed:

```
FD  DISK-FILE
    LABEL RECORDS ARE STANDARD
    VALUE OF FILE-ID IS 'FILE1'.
FD  TAPE-FILE
    LABEL RECORDS ARE STANDARD,
    VALUE OF FILE-ID IS 'FILE2'.
```

  Then an appropriate runtime dialog would be:

```
ENTER FILE ASSIGNMENTS:
>FILE1 = MYUFD>DATA>DISBURSE
>FILE2 = $MT0, S, MYNAME, T1
>/
```

  The first response causes PRIMOS to search a UFD called MYUFD>DATA for a disk file called DISBURSE to use as DISK-FILE in the program.

  You must assign the tape drive (with the PRIMOS ASSIGN statement) before you execute the program. The second response above assumes that a tape drive has been assigned as logical drive 0, with a tape mounted that contains a volume-id of T1 and an owner-id of MYNAME.

- Within the program, a PFMS file named FILEX can be associated with a logical COBOL file named TEST-FILE in either of the following ways.

  1. Value is literal:

```
FD TEST-FILE
   LABEL RECORDS STANDARD
   VALUE OF FILE-ID 'FILEX'.
```

     At execution time, in answer to the -OLD request for file assignments, enter a slash. (For normal I-O, no such action is necessary.)

2. Value is data-name:

```
FD TEST-FILE
   LABEL RECORDS STANDARD
   VALUE OF FILE-ID IS TFILE-NAME.
     .
     .
WORKING-STORAGE SECTION.
77  TFILE-NAME                  PIC X(24).
```

An actual file-name can be associated with the logical file-name TEST-FILE by executing COBOL statements or by adding a VALUE clause to the level-77 description above.

L

Conversion Incompatibilities With Prime's Older COBOL: Rev. 18.4 and Higher

ENVIRONMENT DIVISION

COBOL 74 requires the INPUT-OUTPUT section header if the contents of the section are present.

RECORD KEY names must be unique, independently of qualification.

DATA DIVISION

Files that are produced by the EDITOR or by old COBOL must be identified with the COMPRESSED clause in COBOL 74. Otherwise, the program will appear to run, but the file will not be read correctly.

In COBOL 74, the size of a data item with COMP usage may be 16, 32, or 64 bits, as determined by its PICTURE clause. The default is 16 bits, which is the only option for old COBOL.

In COBOL 74, LOW-VALUES is always treated as alphanumeric.

LOW-VALUES is octal 000 in COBOL 74.

In COBOL 74, only WORKING-STORAGE items with an explicit VALUE clause are initialized.

COBOL 74 inserts filler as needed to make items align on their required boundaries and aligns structures on the largest boundary of any item contained in the structure. See DATA REPRESENTATION AND ALIGNMENT in Chapter 4. A warning message is issued.

An item described with BLANK WHEN ZERO may not have an asterisk in its PICTURE clause in COBOL 74.

LINKAGE SECTION items referenced in the program must appear in the PROCEDURE DIVISION USING header or be subordinate to items in that header.

OTHER

No listing file is created by COBOL 74 unless one is specified on the command line.

COBOL 74 requires -OLD in the compile line for programs written for old COBOL that need file reassignment at runtime.

For continuation of nonnumeric literals, COBOL 74 requires a dash in column 7.

M
Glossary

- Alphabet-name

A programmer-defined word in the SPECIAL-NAMES paragraph of the ENVIRONMENT division that assigns a name to a specific character set or collating sequence.

- Arithmetic Expression (Arith-expr)

A numeric data-name, a numeric literal, such data-names and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. Any arithmetic expression may be preceded by a unary operator.

- Assumed Decimal Point

A decimal point position that does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

- AT END Condition

A condition caused:

1. During the execution of a READ statement for a sequentially accessed file.

2. During the execution of a RETURN statement, when no next logical record exists for the associated sort or merge file.

3. During the execution of a SEARCH statement, when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

- Block

A physical unit of data that is normally composed of one or more logical records. For mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either continued within the block or that overlap the block. The term is synonymous with physical record.

- Called Program

A program that is the object of a CALL statement combined at object time with the calling program to produce a run unit.

- Calling Program

A program that executes a CALL to another program.

- Clause

An ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry, or form a portion of a COBOL procedural statement.

- Collating Sequence

The sequence in which the characters that are acceptable in a computer are ordered for purposes of sorting, merging, and comparing.

- Comment Line

A source program line represented by an asterisk in the indicator area of the line and any characters from the computer's character set in area A and area B of that line. The comment line serves only for documentation in a program. A special form of comment line represented by a slash (/) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line causes page ejection prior to printing the comment.

- Comment-entry

An entry in the IDENTIFICATION division that may be any combination of characters from the computer character set.

- Condition-name

A user-defined word assigned to a specific value, set of values, or range of values, within the complete set of values that a conditional variable may possess; or the user-defined word assigned to a status of an implementor-defined switch or device. Condition-names are defined with level-number 88.

- Conditional Variable

A data item that has one or more values to which a condition-name is assigned.

- Current Record

The record that is available in the record area associated with the file.

- Current Record Pointer

A conceptual entity that is used in the selection of the next record.

- Data-description-entry

An entry in the DATA division that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

- Data-name

A user-defined word that names a data item described in a data-description-entry in the DATA division. A data-name can be subscripted, indexed, or qualified unless these attributes are specifically prohibited by the rules for that format.

- Declaratives

A set of one or more special-purpose sections, written at the beginning of the PROCEDURE division, the first of which is preceded by the keyword DECLARATIVES and the last of which is followed by the keywords END DECLARATIVES. A declarative is composed of a section header, followed by a USE sentence, followed by a set of zero, one, or more associated paragraphs.

- Division Header

A combination of words followed by a period and a space that indicates the beginning of a division. The division headers are:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

- Dynamic Access

An access mode in which records can be obtained from or placed into a mass storage file in a nonsequential manner (see Random Access) and obtained from a file in a sequential manner (see Sequential Access), during the scope of the same OPEN statement.

- Editing Character

A single character or a fixed two-character combination used in a PICTURE clause to change output format. Editing characters are listed in the section PICTURE in Chapter 10.

- Elementary Item

A data item that is described as not being further subdivided.

- File-description-entry

An entry in the FILE section of the DATA division that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

- File-name

A user-defined word that names a file described in a file-description-entry or a sort-merge file-description-entry within the FILE section of the DATA division.

- Imperative Statement

A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement may consist of a sequence of imperative statements.

- Index

1. A computer storage position or register, the contents of which represent the identification of a particular element in a table.

2. A key that identifies a record for the MIDAS+ or old MIDAS utility.

- Index Data Item

A data item in which the value associated with an index-name can be stored.

- Index-name

A user-defined word that names an index associated with a specific table.

- Input Procedure

A set of statements that are executed each time a record is released to a sort file.

- Key

A data item that identifies the location of a record, or a set of data items that serve to identify the ordering of data.

- Key of Reference

The key, either primary or alternate, currently being used to access records within an indexed file.

- Level Indicator

Two alphabetic characters that identify a specific type of file or a position in a hierarchy. A level indicator is found only in the DATA division and must be one of the following: FD, SD.

- Level-number

A user-defined word that indicates the position of a data item in the hierarchical structure of a logical record or that indicates special properties of a data-description-entry. A level-number is expressed as a one- or two-digit number. Level-numbers in the range 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-numbers 66, 77, and 88 identify a data-description-entry with special properties.

- Logical Operator

One of the reserved words AND, OR, or NOT. In the formation of a condition, both or either of AND and OR can be used as logical connectives. NOT can be used for logical negation.

- Merge File

A collection of records to be merged by a MERGE statement. The merge file, identified by SD, is created and can be used only by the merge function.

- MIDAS+ or MIDAS

The Prime utility that handles all indexed and relative file creation and access for COBOL.

- Mnemonic-name

A user-defined word that is associated, in the SPECIAL-NAMES paragraph of the ENVIRONMENT division, with a specified implementor-name, such as CONSOLE or a switch-name.

- Native Character Set

Prime's native character set is the ASCII set defined in Appendix A.

- Noncontiguous Items

Elementary data items, in the WORKING-STORAGE and LINKAGE sections, that bear no hierarchic relationship to other data items.

- Output Procedure

A set of statements to which control is given during execution of a SORT statement after the sort function is completed, or during execution of a MERGE statement after the merge function has selected the next record in merged order.

- Paragraph

In the PROCEDURE division, a paragraph-name followed by a period, a space, and zero, one, or more sentences. In the IDENTIFICATION and ENVIRONMENT divisions, a paragraph header followed by zero, one, or more entries.

- Paragraph Header

A reserved word, followed by a period and a space that indicates the beginning of a paragraph in the IDENTIFICATION and ENVIRONMENT divisions. The permissible paragraph headers are:

```
PROGRAM-ID.
AUTHOR.
REMARKS.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
FILE-CONTROL.
I-O-CONTROL.
```

- Paragraph-name

A user-defined word that identifies and begins a paragraph in the PROCEDURE Division. Paragraph-names must start in columns 8 through 11.

- Pathname

The name of a Prime file, including if necessary its disk name and the name of the UFD and sub-UFDs containing it.

- Phrase

A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL procedural statement or of a COBOL clause.

- Primary Index

For MIDASPLUS, the primary record key or Relative Key.

- Primary Record Key

A key whose contents uniquely identify a record within an indexed file.

- Procedure-name

A paragraph-name in the PROCEDURE division (which may be qualified), or a section-name in the PROCEDURE division.

- Punctuation Character

A character that belongs to the following set:

, ; . " ' ( ) =.

- Qualified Data-name

An identifier that is composed of a data-name followed by OF or IN and another data-name at a higher level of the same hierarchy. The second data-name may also be qualified.

- Qualifier

1. A data-name that is used in a reference together with OF or IN and another data-name at a lower level in the same hierarchy.

2. A section-name that is used in a reference together with OF or IN and a paragraph-name specified in that section.

3. A library-name that is used in a reference together with OF or IN and a text-name associated with that library.

- Random Access

An access mode in which the value of a key data item identifies the record to be accessed in or written to a relative or indexed file.

- Record-description-entry

The total set of data-description-entries associated with a particular record.

- Reserved Word

A COBOL word specified in the list of words in Table A-2 of Appendix A, and which must not appear in the programs as a user-defined word.

- Runfile or Run Unit

The PRIMOS file to be used for execution. It consists of one or more user object files plus any necessary library files.

- Section-name

A user-defined word that names a section in the PROCEDURE Division. Section-names must start within columns 8 through 11.

- Sentence

A sequence of one or more statements, the last of which is terminated by a period followed by a space.

- Sequential Access

An access mode in which logical records are obtained from or placed into a file in a consecutive sequence determined by the order of records in the file.

- Sort File

A collection of records to be sorted by a SORT statement. The sort file, identified by SD, is created and can be used by the sort function only.

- Sort-merge File-description-entry

An entry in the FILE section of the DATA division that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

- Statement

A syntactically valid combination of words and symbols written in the PROCEDURE division, beginning with a verb.

- Subscript

An integer whose value identifies a particular element in a table.

- Table

A set of logically consecutive items, all of the same description, that are defined in the DATA division with the OCCURS clause.

- Table Element

A data item in a set of repeated items comprising a table.

- UFD-name

The name of a PRIMOS user file directory.

- Unary Operator

A plus or a minus sign that precedes a variable or a left parenthesis in an arithmetic expression and that has the effect of multiplying the expression by +1 or -1, respectively.

- Word

1. In storage, 32 bits.

2. A COBOL word is a character-string of not more than 30 characters chosen from the following set of 64 characters:

   0 through 9 (digits)
   A through Z (letters)
   a through z (lowercase letters)
   - (hyphen)
   _ (underscore)

   All words except level-numbers, section-names, segment-numbers, and paragraph-names must contain at least one alphabetic character or a hyphen. A word must not begin or end with a hyphen. It is delimited by a space, or by proper punctuation. A word may contain more than one embedded hyphen; consecutive embedded hyphens are also permitted.

INDEX

Index

-64V compile option 2-11

== (pseudo-text delimiter) 4-9, 8-18

Abbreviated combined conditions 4-44, 4-45

ACCEPT 8-8 to 8-10

ACCESS MODE:
  general rules 12-7, 12-8
  indexed files 12-7, 12-8
  relative files 13-3, 13-7, 13-8

ADD 8-11, 8-12

Algebraic signs 4-31

Alignment rules:
  Prime extension 4-28 to 4-31

ALL literal 4-14

-ALLERRORS compile option 2-7

Alphabet-name M-1

Alphabetic item 4-20

Alphanumeric edited item 4-20, 7-28

Alphanumeric item 4-20, 7-29

ALTER 8-13

ALTERNATE RECORD KEY:
  Prime extension 12-9
  rules 12-9

ANSI standard:
  definition 1-1
  levels of COBOL C-1, C-2
  Prime extension 1-1, 1-2, I-2 to I-4
  Prime support I-1, I-2

Area A, Area B 4-7, 4-8

Arithmetic expressions:
  definition 4-35, M-1
  subscripts containing 10-20
  table of symbol combinations 4-35
  with SET 10-8

Arithmetic operators 4-36, 4-37

Arithmetic statements 4-37, 4-38, 8-3 to 8-6

Arithmetic symbols A-3

ASCENDING KEY:
description 10-2 to 10-5
MERGE 8-47

ASCII character set and collating sequence A-10 to A-14

ASSIGN 6-7 to 6-9, 12-7

Assumed decimal point M-1

Asterisk:
comment line 4-7
PICTURE symbol 7-30

AT END condition:
definition M-2
indexed files 12-4
relative files 13-4, 13-18

Batch environment 1-5

-BINARY compile option 2-10

Binary item 4-23, 4-24

BLANK WHEN ZERO 7-23

Block M-2

BLOCK CONTAINS 7-4, 7-7, 14-12

Blocking records 14-1, 14-12, 14-13

CALL:
format 9-5
rules 8-1, 8-2, 8-13, 8-14, 9-5, 9-6

Called program M-2

Calling program M-2

Calling programs in other languages 9-1, 9-15, 9-16

Carriage control, integer values 8-102

Categories of data 4-20, 4-21

CBLLIB:
library files list F-1 to F-2
overview 3-2 to 3-3
use in loading 3-2, 3-3

CBLSW 0 through 7 6-5

Character set:
ASCII A-9 to A-14
collating sequence 4-11
Prime COBOL 74 4-9, 4-10, 4-10
Prime extensions 4-11

Character strings 4-11

Class condition 4-40, 4-41

Classes of data 4-20, 4-21

Clause M-2

CLOSE:
format 8-14
indexed files 12-12
relative files 13-12
tape processing 14-17

COBOL 74:
library (CBLLIB) 3-2, 3-3
library files list F-1 to F-2
PRIME's, overview 1-2, 1-3
reserved words A-6 to A-8
under PRIMOS 1-2, 1-3

CODE-SET 7-4, 7-8, 14-14

Coding:
column numbers 4-8
rules 4-7, 4-8
symbols A-2

Collating sequence 4-11, M-2

COLLATING SEQUENCE IS 11-6 to 11-8

Combined and negated combined conditions 4-43

Comma 4-8

Command line options 2-2 to 2-13

Comment line M-3

Comment-entry 5-2, M-3

COMP 4-23, 4-24

COMP-1 4-26 to 4-28

COMP-2 4-26 to 4-28

COMP-3 4-23

Comparisons (See IF)

Compiler-directing statements, overview and verbs 8-3

Compiler:
- command line format 2-1
- command line options 2-2 to 2-13
- error messages 2-17
- table of command line options 2-3, 2-4
- table of output options 2-16

Complex conditions 4-42

Composite of operands 8-4

COMPRESSED/UNCOMPRESSED 7-4 to 7-6

Compression control 7-6

COMPUTATIONAL 4-23, 4-24

COMPUTATIONAL-1 4-26 to 4-28

COMPUTATIONAL-2 4-26 to 4-28

COMPUTATIONAL-3 4-23

COMPUTE 8-16, 8-17

COMPUTE CORRESPONDING 8-16, 8-17

Condition evaluation rules 4-45 to 4-47

Condition symbols A-3

Condition-name:
- condition 4-41
- definition M-3
- overview 4-16
- with level 88 7-19, 7-20

Conditional expressions:
- abbreviated combined conditions 4-44, 4-45
- class condition 4-40, 4-41
- combined and negated combined conditions 4-43
- comparison of operands 4-37, 4-38
- complex conditions 4-42
- condition evaluation rules 4-45 to 4-47
- condition-name condition 4-41
- definition 4-38
- multiple conditions 4-44
- negated simple conditions 4-42
- relation condition 4-38 to 4-40
- sign condition 4-42
- switch-status condition 4-41

Conditional statements 8-3

Conditional variable 4-16, 7-20, 8-38, M-3

CONFIGURATION section:
- description 6-1, 6-2
- overview 4-2

Connectives 4-13

CONSOLE IS 6-4

Continuation of literals 4-19

Conventions of notation:
  COBOL 4-5, 4-6
  Prime xiv, xv

Conversion from old to new Prime COBOL L-1

COPY 8-18 to 8-20

CORRESPONDING, rules 8-6

CREATK:
  data subfile questions E-4
  dialogs E-2
  example, indexed files E-6
  example, relative files E-12, E-13
  indexed files E-3 to E-5
  relative (direct access) files E-11 to E-13

CURRENCY SIGN 6-4

Currency symbol 7-30

Current record pointer:
  definition M-3
  indexed files 12-3
  overview 8-56
  relative files 13-3
  sequential files 8-70

Current record:
  definition M-3

Data alignment 4-28 to 4-31

Data categories:
  characteristics 4-20, 4-21
  table of classes and categories 4-21

Data classes 4-20, 4-21

DATA division:
  description 7-1
  example 7-52 to 7-54
  format 7-2
  indexed sequential files 12-11
  overview 4-2
  relative files 13-11
  tape files 14-11 to 14-16

Data levels:
  elementary item 4-19
  group item 4-19

DATA RECORD 7-4, 7-5, 7-9, 11-5

Data representation and alignment 4-22 to 4-31

Data type compatibility with other languages 9-15, 9-16

Data-description-entry M-3

Data-name:
  definition M-4
  formation 4-15, 4-16
  overview 4-16
  subscripting 10-19, 10-20
  usage 7-22

DATE-COMPILED 5-2 to 5-3

-DEBUG compile option 2-11

Debugger:
  compiling and loading D-1, D-2
  examples D-3 to D-4
  overview 1-6, D-1
  special definitions D-2, D-3

Decimal data type (overpunch symbols) A-23

DECIMAL-POINT 6-4

Declaratives M-4

DECLARATIVES section:
  format 8-1, 8-2
  overview 4-2
  rules 8-7

Default loading 3-1 to 3-3

Defaults:
  ANSI notation 4-5, 4-6
  Prime notation xiv

DELETE:
format 8-21, 12-13
rules 12-13, 13-13

DESCENDING KEY:
description 10-2 to 10-5
MERGE 8-47

Device-names, table of 6-8

Diagnostics (See Error messages)

-DIAGSONLY compile option 2-11

Direct-access files (See Relative files)

DISPLAY:
format 4-22, 7-43, 7-44
rules 8-22, 8-23

DIVIDE 8-24 to 8-26

Division header M-4

Divisions, COBOL program 4-1, 4-2

Double-precision floating-point item 4-26 to 4-28

Dynamic access M-4

EBCDIC character set and collating sequence A-15, A-16

Edit symbols in PICTURE clauses A-4, A-5

Editing character M-4

Editing rules in PICTURE clauses 7-31 to 7-35

EDITOR, description 1-6

EJECT 8-27

Elementary item 4-19, M-4

Embedded signs A-23

END DECLARATIVES:
format 8-1
rules 8-2, 8-6

ENTER 8-28, 9-7

ENVIRONMENT division:
example 6-12
format 6-1
overview 4-1
sort-merge module 11-2, 11-3

Error messages:
COBOL runtime error messages B-2
compile time error messages 2-17, B-1, B-2
LOAD 3-5
LOAD (interdependent programs) 9-11
MIDASPLUS runtime error messages B-2, B-3
PRIMOS error messages B-3
System runtime error messages 3-7
tape files 14-23 to 14-25

-ERRORFILE compile option 2-7

Executing programs:
interdependent programs 9-10
overview 3-5, 3-6

EXHIBIT 8-29

EXIT 8-30

EXIT PROGRAM 8-31, 9-8

-EXPLIST compile option 2-5

Exponentiation 4-26 to 4-28

FD (File Description) 7-4

Federal Information Processing (FIPS) 2-7, C-1, C-2

Figurative constants 4-13, 4-14

File assignment for tape:
assignment error messages 14-6, 14-7

examples, -OLD 14-6, K-3
tape file assignment format K-4, 14-5
with -OLD 14-4, K-3
with normal I-O 14-4

File assignment:
at runtime K-1 to K-4
error messages 3-8, 3-9
in DATA division 7-13 to 7-15
with -OLD K-1 to K-4

File Description (FD):
format 7-4
rules 7-3 to 7-5

FILE section:
format 7-3
function 7-3
overview 4-2
sort file description 11-5

File status codes:
all files A-17 to A-19
Prime-defined codes A-18, A-19
with indexed files 12-3, 12-4
with relative files 13-3

FILE STATUS:
indexed files 12-3, 12-4
Prime extension 12-9
relative files 13-3, 13-4
rules 12-9

File unit numbers, table 2-14

FILE-CONTROL:
format 6-7 to 6-9
indexed files 12-7 to 12-9
Prime extension, relative files 13-9
relative files 13-7 to 13-9
tape files 14-9, 14-10

File-description-entry M-4

File-names:
definition M-5
normal naming 2-14
older naming convention 2-15
PRIMOS default names 2-15
rules 4-16
table of default UFD's 2-15

FILLER, rules 7-22

-FIPS compile options 2-7, C-1

FIPS levels C-1, C-2

Fixed insertion 7-31, 7-32

Floating insertion 7-31 to 7-34

Floating-point data type 4-26 to 4-28

-FORCEBINARY compile topion 2-11

Format notation:
braces 4-6
brackets 4-6
ellipsis 4-6
examples 4-6
level-numbers 4-5
list of ANSI notation 4-5, 4-6
punctuation 4-6
special characters 4-6

FORMS Management System 1-7

FORMS, description of 1-7

GIVING:
overview 8-4
rules 11-22, 11-23

GO TO 8-32, 8-33

GO TO DEPENDING 8-32, 8-33

GOBACK 8-34, 9-9

Group item 4-19

-HELP compile option 2-13

-HEXADDRESS compile option 2-5

Hexadecimal addition table A-22

Hexadecimal and decimal conversion A-21

HIGH-VALUE 4-14

I-O-CONTROL:
format 6-10, 6-11
indexed files 12-10
relative files 13-10
sort-merge program rules 11-3, 11-4
tape files 14-9

ID DIVISION (See Identification Division)

IDENTIFICATION division:
description 5-1
example 5-4
overview 4-1
Prime extensions 5-1 to 5-3

IF 8-35 to 8-38

IF CORR (See IF CORRESPONDING)

IF CORRESPONDING 8-35, 8-37

Imperative statements 8-3, M-5

Implementor-names 4-15

Incompatibilities between Prime's COBOLs L-1

Index M-5

Index data item 4-35, M-5, 10-6

Index item 4-24, 10-6

INDEX usage 4-24, 4-25, 7-43

Index-name M-5

INDEXED BY:
format 10-1, 10-7
general rules 10-2 to 10-4, 10-7

Indexed I-O module:
MIDASPLUS index 12-1
overview 12-1

Indexed sequential files:
access modes 12-3
error handling 12-6
file handling 12-5, 12-6
loading and executing programs 12-1
organization 12-2
overview 12-1
primary and secondary keys 12-2, 12-3
record handling 12-6
sample program 12-28 to 12-32

Indexing:
direct 4-34, 10-18
formats 4-34, 10-17
relative 4-34, 10-18
rules 4-34, 4-35

-INPUT compile option 2-2

Input procedure M-5

INPUT PROCEDURE IS 11-20, 11-21

INPUT-OUTPUT section:
format 6-7
overview 4-2

Input/Output handling, old COBOL 2-13

Input/Output statements, permissible A-20

INSPECT 8-38 to 8-46

Interactive environment 1-4, 1-5

Interprogram communication:
LINKAGE section 9-2, 9-3
overview 9-1
rules 9-2, 9-3
sample program 9-12 to 9-14

INVALID KEY condition:
indexed files 12-4
relative files 13-4

JUSTIFIED 7-24

KBUILD utility:
  example, indexed files E-9 to E-10
  example, relative file E-15
  functions E-3
  indexed files E-8, E-9
  overview E-3
  relative files E-12, E-14

Key M-5

KEY data-names 11-7, 11-17, 11-18

Key of reference M-5

Keyed-index files, (See MIDASPLUS)

Keywords 4-13

LABEL RECORDS:
  general rules 7-4, 7-10

LABEL utility:
  errors using LABEL 14-23 to 14-25
  -HELP command 14-25
  overview 14-22
  using LABEL 14-22, 14-23

Language interfaces, Prime
  high-level languages 1-7

Level indicator M-6

Level-number:
  01 rules 7-17 to 7-19
  66 rules 7-17, 7-20, 7-38, 7-39
  77 rules 7-17 to 7-19, 7-48 to 7-51
  88 rules 7-17 to 7-20, 7-45, 7-46
  overview 4-5, 4-15, 4-15, 7-18 to 7-21, M-6

LI (load utility command) 3-2

Libraries 1-6

LIBRARY (load utility command) 3-2

Library Module (See COPY)

LINKAGE section:
  format 7-50
  overview 4-2
  rules 7-50, 7-51, 8-2
  syntax rules 9-2, 9-3

Linkage segment:
  increase work space in 2-13

-LISTING compile option 2-5

Listing file:
  contents 2-5
  example 4-4, 4-5

Literals:
  continuation of 4-19
  definition 4-17
  nonnumeric literals 4-17, 4-18
  numeric literals 4-18

-LNKWRK compile option 2-13

LOAD (SEG command) 3-1

Load and execute utility (See SEG utility)

Loading programs:
  error messages 3-5
  examples 3-1 to 3-5
  interdependent programs 9-10
  SORT or MERGE 11-2, 11-26

Logical operator 4-42, M-6

LOW-VALUE 4-14

-MAP compile option:
  example G-1 to G-9
  overview 2-6
  showing alignment 4-30

-MAPWIDE compile option:
  overview 2-6

Merge file M-6

MERGE:
  definition M-6
  format 8-47

rules 11-6 to 11-9
sample program 11-9 to 11-11

MIDAS (See MIDASPLUS)

MIDASPLUS:
building a template E-1
definitions E-1, E-2, M-7
description 1-6, 1-7

Mnemonic-name:
definition M-6
overview 4-16, 6-4

MOVE 8-48 to 8-51

MOVE CORR (See MOVE CORRESPONDING)

MOVE CORRESPONDING 8-48, 8-50

Multidimensional tables 10-20

Multiple conditions 4-44

MULTIPLY 8-52, 8-53

MULTIPLY CORR (See MULTIPLY CORRESPONDING)

MULTIPLY CORRESPONDING 8-50, 8-53

Multivolume tape files 14-7

Native character set M-6

NCBLLIB 3-2

NCBLLIB:
library files list F-1 to F-2

Negated simple conditions 4-42

Nested IF:
definition 8-36
structure 8-37

NEXT SENTENCE 8-35, 8-36

Nines syndrome 4-28

-NOBINARY compile option 2-12

-NOCALCINDEX compile option 2-11

Noncontiguous data items 7-48, 7-49, M-7

-NOOPTIMIZE compile option 2-12

-NOOWNERID compile option 2-12

-NOSYNTAXMSG compile option 2-7

NOTE 8-54

-NOTTYDIAGS 2-7

-NOTTYDIAGS compile option 2-7

Numeric edited item 4-20, 7-29

Numeric item 4-20, 7-29

Object code:
augmented 2-11 to 2-13
existence and properties of 2-10, 2-11

Object file, creation of 2-10 to 2-13

OBJECT-COMPUTER 6-1, 6-3

OCCURS:
description 7-25
formats 10-2
general rules 10-4, 10-5
syntax rules 10-2, 10-3

Octal and decimal conversion A-22

OFF STATUS 6-4, 6-5

-OFFSET compile option 2-6

-OLD compile option 2-13, L-1

ON SIZE ERROR 8-5

ON STATUS 6-4, 6-5

OPEN:
format and rules 8-55 to 8-57
indexed files 12-14
relative files 13-14, 13-15
tape processing 14-18, 14-19

Operand combinations with SET 10-9

Operands:
comparison of 4-38 to 4-40
composite of 4-38, 8-4
floating point 4-26 to 4-28
overlapping 4-38

-OPTIMIZE compile option 2-12

ORGANIZATION IS INDEXED 12-7

Output procedure M-7

OUTPUT PROCEDURE IS:
format 8-47
rules 11-22, 11-23

Overpunch A-23

OWNER IS 7-4, 7-11

Packed decimal item 4-23

Paragraph M-7

Paragraph header M-7

Paragraph-name 4-17, M-8

Parentheses 4-8

Parity bit A-9, A-14

Pathname M-7

PERFORM 8-58 to 8-67

Period as separator 4-8

PFMS 6-8

Phantom environment 1-5

Phrase M-8

PIC 7-16, 7-26 to 7-30

PICTURE character string 7-26

Picture strings:
definition 4-11
rules 7-26

PICTURE:
alphabetic item 7-26, 7-31
alphanumeric edited item 7-28, 7-31
data categories 7-26, 7-28
editing rules 7-31 to 7-35
examples 7-27
function and rules 7-26
numeric edited item 7-28, 7-31
numeric item 7-28, 7-31
size 7-28
symbols 7-28 to 7-30

Primary index M-8

Primary record key M-8

Prime COBOL 74:
character set 4-9, 4-10
features available I-1, I-2
high and low values J-2
implementation-dependent features, Rev. 18 J-1, J-2
maximum numbers J-2
maximum sizes J-1
system resources for 1-5 to 1-7

Prime extensions:
ACCESS MODE IS 12-27
alignment of substructures 4-29, 4-30
ALTERNATE RECORD KEY 12-9
arithmetic expressions with SET 8-81, 10-8
CALL 9-5
character set 4-11
COMP-3 4-23
COMPRESSED/UNCOMPRESSED 7-5, 7-6
COMPUTE CORRESPONDING 8-16, 8-17
computer-name 6-2

condition-name 6-5
CONFIGURATION section 6-2
CORRESPONDING 8-6
cross-reference listing 2-7, H-1 to H-7
debugger D-1 to D-4
device-names 6-8
DISPLAY...NO ADVANCING 8-22
DIVIDE 8-25
EJECT 8-27
EXHIBIT NAMED 9-28
FD and SD, order of 7-2
FILE STATUS 12-9
file status codes A-17, A-18
FILLER 7-22
FIPS checking 2-7, C-1, C-2
floating-point item (COMP-1, COMP-2) 4-26 to 4-28
GO TO 8-32
GOBACK 8-33, 9-9
I-O-CONTROL 6-2
IF CORRESPONDING 8-35, 8-37
IF...THEN...OTHERWISE 8-35
LABEL RECORD 7-10
level-number 7-18, 7-19, 8-2
lowercase letters 4-11
-MAP listing G-1 to G-9
MOVE 8-48
MULTIPLY CORRESPONDING 8-53
NOTE 8-54
overview 1-1, 1-2
READY TRACE 8-71, 8-72
RECORD KEY 12-8
REDEFINES 7-36
relative files, FILE CONTROL paragraph 13-9
REMARKS 5-2, 5-3
RESET TRACE 8-74
REWRITE 8-77, 12-19, 12-19, 13-19
SEEK 8-80, 8-80, 12-21, 13-21
single quote 4-11, I-2
SKIP 8-82
SOURCE-COMPUTER, OBJECT-COMPUTER 6-2, 6-3
subscript range checking 2-12
subscripting levels 10-20
subscripts containing arithmetic expressions 4-33, 10-20
to ANSI notation 4-7
to ANSI standard, list I-2 to I-4
to IDENTIFICATION division 5-1 to 5-3
underscore in names 4-11
USAGE 7-43
USING 8-2
word formation 4-12
WRITE 8-101, 12-26, 13-25, 13-25

Prime restrictions:
computer-name 6-2
RECORD KEY 12-9
STRING 8-88
UNSTRING 8-96

PRIMOS:
compatibility of old and new Prime COBOL 1-4
definition of 1-2
operating environment 1-4
PRIMOS-level commands 1-2, 1-3

Procedural sections, overview 4-2

PROCEDURE division:
arithmetic statements in 8-3 to 8-6
example 8-104 to 8-109
formats 8-1, 8-2
interprogram communication 9-4
maximum size J-1
overview 4-2
relative files 13-12
syntax rules 8-1 to 8-3

Procedure statements 8-7

Procedure-name M-8

-PRODUCTION compile option 2-12

Program environments, under PRIMOS 1-4, 1-5

Program outline and example 4-3 to 4-5

Program statistics 2-8, 2-9

PROGRAM-ID 5-2

Programmer-defined words 4-15 to 4-17

Pseudo-text 4-9, 8-18 to 8-20

Punctuation:
  characters 4-8, 4-9, M-8
  rules 4-8, 4-9
  symbols list A-2

Qualification of names 4-31 to 4-33

Qualified data-name M-8

Qualifier M-9

Quotation mark 4-8

QUOTES 4-14

Random access M-8

-RANGE compile option 2-12

READ:
  indexed files 12-15 to 12-18
  relative files 13-16 to 13-18
  sequential files 8-68 to 8-70
  tape processing 14-20

READY TRACE 8-71, 8-72

Receiving items 8-87, 8-93

RECORD CONTAINS 7-4, 7-12, 11-5

Record description:
  definition M-9
  format 7-16, 7-17
  indexed files 12-11
  rules 7-17

RECORD KEY:
  general rules 12-8, 12-9
  Prime extension 12-8
  Prime restriction 12-9

REDEFINES:
  example 7-37
  rules 7-36, 7-37

Reference tables, list of A-1

Relation condition 4-38 to 4-40

Relative files:
  (See also MIDASPLUS)
  error handling 13-6
  file handling 13-5
  overview 13-1, 13-2
  record handling 13-5, 13-6
  RELATIVE KEY 13-1
  sample program 13-27 to 13-31

Relative I-O module 13-1, 13-2

RELEASE:
  format 8-73
  general rules 11-13, 11-14
  sample program 11-14

RENAMES 7-17, 7-38, 7-39

Repeat integer 7-28

RERUN 6-10, 12-10, 13-10, 14-19

Reserved words:
  connectives 4-13
  definition M-9
  figurative constants 4-13, 4-14
  implementor-names 4-15
  keywords 4-13
  list A-6 to A-8
  optional words 4-13
  special-character words 4-14

RESET TRACE 8-74

RETURN 8-75, 11-15, 11-16

REWRITE:
  format and rules 8-76, 8-77
  indexed files 12-19, 12-20
  Prime extension 12-19, 13-19
  relative files 13-19, 13-20

-RMARGIN compile option 2-5

ROUNDED:
  examples of results 8-5
  rules 8-4

Run unit M-9

Runfile M-9

SAME RECORD AREA 12-10

Sample programs:
ACCEPT 8-10
CREATK E-6, E-7, E-12, E-13
DATA division 7-52 to 7-54
ENVIRONMENT division 6-12
general outline 4-4 to 4-5
IDENTIFICATION division 5-4
indexed file 12-28 to 12-32
interprogram communication 9-12 to 9-14
KBUILD E-9, E-10, E-15
-MAP option G-3 to G-9
MERGE 11-9 to 11-12
PROCEDURE division 8-104 to 8-109
relative file 13-27 to 13-31
RELEASE 11-14
SORT 11-24 to 11-26
START 12-23, 12-24
STRING 8-88, 8-89
table handling 10-21 to 10-27
tape file 14-26 to 14-29
UNSTRING 8-10, 8-96
-XREFSORT option H-2 to H-7

SD file-name 11-5

SEARCH ALL:
format 8-78, 8-79, 10-10
rules 10-11 to 10-14

SEARCH:
example 10-12
flowchart 10-15
format 8-78, 8-79, 10-10
rules 10-11 to 10-14

Section-names 4-17, M-9

Section:
format 8-1
rules 4-17, 8-2

SEEK 8-80, 12-21, 13-21

SEG (load utility):
definition 1-6
overview 3-1 to 3-3

Segment-number 4-17, 8-2

SELECT 6-7 to 6-9, 12-7

Semicolon 4-8

Sending items 8-87, 8-93

Sentence M-9

Separators 4-8

Sequential access M-9

Sequential files:
CLOSE 8-14
DELETE 8-21
OPEN 8-55
READ 8-68 to 8-70
REWRITE: 8-76
WRITE 8-100

SET:
format 8-81, 10-8
rules 10-8, 10-9

Sign condition 4-42

Sign symbols and unary operators A-3

SIGN:
representation 7-41
rules 7-40, 7-41

-SIGNALERRORS compile option 2-13

-SILENT compile option 2-7

-SILENT2 compile option 2-7

-SILENT3 compile option 2-8

Simple conditions 4-38

Simple insertion 7-31

Single-precision floating-point item 4-26, 4-28

SIZE ERROR 8-5

SKIP 8-82

-SLACKBYTES compile option 2-8, 4-30

Sort file M-9

Sort-merge file-description-entry M-9

Sort-merge module:
  definition 11-1
  strategy 11-2

SORT:
  format 8-83, 11-15
  general rules 11-19, 11-20
  sample program 11-24 to 11-26
  strategy 11-1
  syntax rules 11-18, 11-19

-SOURCE compile option 2-2

Source file 2-2

Source listing:
  existence and contents of 2-5
  options 2-2, 2-5

SOURCE-COMPUTER 6-1, 6-2

Space as separator 4-8, 4-9

SPACES 4-14

Special insertion 7-31, 7-32

SPECIAL-NAMES 6-1, 6-4 to 6-6

START:
  format 8-84, 12-22, 13-22
  indexed files 12-22 to 12-24
  program sample 12-23, 12-24
  relative files 13-22, 13-23

Statement definition 8-3, M-9

-STATISTICS compile option 2-9

STOP 8-85

STRING:
  format 8-86
  rules 8-86 to 8-88
  sample program 8-88, 8-89

Subscript M-10

Subscripting:
  data-names 10-19, 10-20
  format 10-18
  general rules 4-33, 10-19
  literals 10-19

SUBTRACT 8-90, 8-91

Suppression editing 7-34, 7-35

Switch settings 3-6

Switch-names 6-5

Switch-status condition 4-41

SYNC 7-42

SYNCHRONIZED 7-42

Table M-10

Table element M-10

Table handling:
  example of strategy 10-16, 10-17
  overview 10-1
  sample program 10-21 to 10-27

Tables, multidimensional 10-20

Tape file assignments, (See File assignment for tape)

Tape file-name format K-3

Tape processing:
  blocking 14-1
  DATA division 14-11 to 14-16
  ENVIRONMENT division 14-9, 14-10
  HELP facility of LABEL 14-25
  multivolume 14-7
  nonstandard labels 14-17
  normal loading 14-2
  older loading procedure 14-3
  PROCEDURE division 14-17 to 14-21
  sample program 14-26 to 14-29

tape drive assignments 14-3
tape structure 14-1

-TOTALS compile option 2-10

-TRUNCDIAGS compile option 2-10

UFD-name M-10

Unary operator M-10

Unpacked decimal item 4-22

UNSTRING:
format 8-92
rules 8-92 to 8-96
sample program 8-9, 8-96, 8-97

USAGE:
COMP 4-23, 4-24, 7-43, 7-44
COMP-1 4-26 to 4-28
COMP-2 4-26 to 4-28
COMP-3 4-23
COMPUTATIONAL 4-23, 4-24, 7-43, 7-44
COMPUTATIONAL-1 4-26 to 4-28
COMPUTATIONAL-2 4-26 to 4-28
COMPUTATIONAL-3 4-23
data types 4-22
DISPLAY 4-22, 7-43, 7-44
INDEX 7-43, 10-6
index item 10-6
rules 7-43, 7-44

USE:
format 8-98
rules 8-98, 8-99

USING:
formats 8-1
rules 8-1, 8-2, 11-20 to 11-21

VALUE 7-45 to 7-47

VALUE OF FILE-ID 7-4, 7-13 to 7-15, 14-15, 14-16

Word M-10

Word formation 4-12, M-10

WORKING-STORAGE SECTION:
format 7-48
overview 4-2, 7-1
rules 7-17, 7-48, 7-49

WRITE:
indexed files 12-26
Prime extension 12-26, 13-25
record keys 13-26
relative files 13-25
rules 12-26, 12-27
sequential files 8-100 to 8-102
tape processing 14-21

-XREF compile option:
overview 2-6

-XREFSORT compile option:
overview 2-6
sample program H-1 to H-7

Zero suppression 7-31, 7-34, 7-35

ZEROS 4-14

READER RESPONSE FORM

DOC5039-184 COBOL 74 Reference Guide

Your feedback will help us continue to improve the quality, accuracy, and organization of our user publications.

1. How do you rate the document for overall usefulness?

   ___excellent ___very good ___good ___fair ___poor

2. Please rate the document in the following areas:

   Readability: ___hard to understand ___average ___very clear

   Technical level: ___too simple ___about right ___too technical

   Technical accuracy: ___poor ___average ___very good

   Examples: ___too many ___about right ___too few

   Illustrations: ___too many ___about right ___too few

3. What features did you find most useful? _____

   _____

   _____

   _____

4. What faults or errors gave you problems? _____

   _____

   _____

   _____

Would you like to be on a mailing list for Prime's current documentation catalog and ordering information? ___yes ___no

Name: _____ Position: _____

Company: _____

Address: _____

_____ Zip: _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

First Class Permit #531 Natick, Massachusetts 01760

BUSINESS REPLY MAIL

Postage will be paid by:

PRIME

Attention: Technical Publications
Bldg 10B
Prime Park, Natick, Ma. 01760