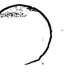
Data General

Customer Documentation

Interactive COBOL Programmer's Reference



Interactive COBOL Programmer's Reference

093-705013-05

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-705013
Copyright © Data General Corporation, 1982, 1983, 1984, 1989, 1990, 1991
All Rights Reserved
Unpublished — All rights reserved under the Copyright laws of the United States
Printed in the United States of America
Rev. 05, April 1991
Licensed Material — Property of Data General Corporation

Notice

DATA GENERAL CORPORATION (DGC) HAS PREPARED THIS DOCUMENT FOR USE BY DGC PERSONNEL, LICENSEES, AND CUSTOMERS. THE INFORMATION CONTAINED HEREIN IS THE PROPERTY OF DGC; AND THE CONTENTS OF THIS MANUAL SHALL NOT BE REPRODUCED IN WHOLE OR IN PART NOR USED OTHER THAN AS ALLOWED IN THE DGC LICENSE AGREEMENT.

DGC reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult DGC to determine whether any such changes have been made.

THE TERMS AND CONDITIONS GOVERNING THE SALE OF DGC HARDWARE PRODUCTS AND THE LICENSING OF DGC SOFTWARE CONSIST SOLELY OF THOSE SET FORTH IN THE WRITTEN CONTRACTS BETWEEN DGC AND ITS CUSTOMERS. NO REPRESENTATION OR OTHER AFFIRMATION OF FACT CONTAINED IN THIS DOCUMENT INCLUDING BUT NOT LIMITED TO STATEMENTS REGARDING CAPACITY, RESPONSE-TIME PERFORMANCE, SUITABILITY FOR USE OR PERFORMANCE OF PRODUCTS DESCRIBED HEREIN SHALL BE DEEMED TO BE A WARRANTY BY DGC FOR ANY PURPOSE, OR GIVE RISE TO ANY LIABILITY OF DGC WHATSOEVER.

This software is made available solely pursuant to the terms of a DGC license agreement, which governs its use.

AViiON, CEO, DASHER, DATAPREP, DESKTOP GENERATION, ECLIPSE, ECLIPSE MV/4000, ECLIPSE MV/6000, ECLIPSE MV/8000, GENAP, INFOS, microNOVA, NOVA, PRESENT, PROXI, SWAT, and TRENDVIEW are U.S. registered trademarks of Data General Corporation; and AOSMAGIC, AOS/VSMAGIC, AROSE/PC, ArrayPlus, BaseLink, BusiGEN, BusiPEN, BusiTEXT, CEO Connection, CEO Connection/LAN, CEO Drawing Board, CEO DXA, CEO Light, CEO MAILI, CEO Object Office, CEO PXA, CEO Wordview, CEOwrite, COBOL/SMART, COMPUCALC, CSMAGIC, DASHER/One, DASHER/286, DASHER/286-12c, DASHER/286-12j, DASHER/386, DASHER/386-16c, DASHER/386-25, DASHER/386-25k, DASHER/386sx, DASHER/386SX-16, DASHER/486-25, DASHER/LN, DATA GENERAL/One, DESKTOP/UX, DG/500, DG/AROSE, DGConnect, DG/DBUS, DG/Fontstyles, DG/GATE, DG/GEO, DG/HEO, DG/L, DG/LIBRARY, DG/UX, DG/XAP, ECLIPSE MV/1000, ECLIPSE MV/1400, ECLIPSE MV/2000, ECLIPSE MV/2500, ECLIPSE MV/3500, ECLIPSE MV/5000, ECLIPSE MV/5500, ECLIPSE MV/7800, ECLIPSE MV/9500, ECLIPSE MV/10000, ECLIPSE MV/15000, ECLIPSE MV/18000, ECLIPSE MV/20000, ECLIPSE MV/30000, ECLIPSE MV/40000, FORMA-TEXT, GATEKEEPER, GDC/1000, GDC/2400, Intellibook, microECLIPSE, microMV, MV/UX, PC Liaison, RASS, REV-UP, SLATE, SPARE MAIL, SUPPORT MANAGER, TEO, TEO/3D, TEO/Electronics, TURBO/4, UNITE, WALKABOUT, WALKABOUT/SX, and XODIAC are trademarks of Data General Corporation.

UNIX is a U.S. registered trademark of American Telephone and Telegraph Company. MS-DOS is a U.S. registered trademark of Microsoft Corporation. 386/ix is a trademark of Interactive Systems Corporation.

Restricted Rights Legend: Use, duplication, or disclosure by the U. S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at [FAR] 52.227-7013 (May 1987).

Data General Corporation
4400 Computer Drive
Westboro, MA 01580

Interactive COBOL Programmer's Reference
093-705013-05

Revision History:		Effective with:
Original Release	– June 1982	
First Revision	– April 1983	
Second Revision	– August 1984	
Third Revision	– November 1989	
Fourth Revision	– June 1990	
Fifth Revision	– April 1991	Interactive COBOL Rev. 1.70

A vertical bar in the margin of a page indicates substantive technical change from the previous revision.

Preface

Document Set

Interactive COBOL is documented by a set of manuals that describe the language, its utilities, and the system-dependent features that affect its use. The *Interactive COBOL Programmer's Reference* defines the Interactive COBOL programming language. It is your primary reference regardless of your operating system.

Each manual set also includes a user's guide and a utilities manual that explain the features of your particular operating system as they relate to Interactive COBOL. The system-dependent user's guides describe the file system and give specific instructions for invoking the runtime system, compiler, and debugger, if it exists for that operating system. The system–dependent utilities manual explains how to use the Interactive COBOL utilities on that operating system.

The titles and order numbers of Interactive COBOL documents referred to in this manual are listed in "Related Documents" at the end of the manual.

Scope

Chapters 1-4 of this manual explain COBOL concepts and present an overview of Interactive COBOL, file organization and access, screen management, and source entry. Chapters 5-8 discuss Interactive COBOL syntax and list syntax formats.

Since there is no significant difference to the programmer, this manual uses the term DG/RDOS to refer to RDOS and DG/RDOS, and AOS/VS to refer to AOS/VS, and AOS/VS II.

UNIX® is a registered trademark of AT&T. However, since the DG/UX™ software product and the Interactive UNIX Systems software product have been derived from, or relate to, or work in conjunction with UNIX software, these two software products are sometimes referred to herein as UNIX products, solely for the purpose of improved readability. This liberty is taken in this manual only where Interactive UNIX Systems and UNIX, or DG/UX and UNIX, have areas of commonality or overlap, and not where the differences between them are significant.

Organization

This manual is divided into eight chapters, two appendixes, and a glossary.

Chapter 1 presents an overview of language, data, and Procedure Division concepts.

Chapter 2 describes COBOL file organization and access.

Chapter 3 explains interactive screen management.

Chapter 4 discusses COBOL source entry formats: CRT and card.

Chapter 5 discusses the Identification Division.

Chapter 6 discusses the Environment Division.

Chapter 7 discusses the Data Division, including the Screen Section, an Interactive COBOL extension.

Chapter 8 lists Interactive COBOL verbs in alphabetical order, with formats, examples, and rules for use.

Appendix A lists the 7-bit and 8-bit ASCII character sets.

Appendix B lists the ANSI Standard and Interactive COBOL reserved words.

Notational Conventions

The conventions described below are used in this manual to represent the various elements of COBOL language syntax. The following example contains most of the syntactical elements used:

$$\underline{\text{ACCEPT}}\ \textit{screen-name} \left[\text{AT} \left[\underline{\text{LINE}}\ \textit{id-lit} \right] \left[\left\{ \begin{array}{l} \underline{\text{COLUMN}} \\ \underline{\text{COL}} \end{array} \right\} \textit{id-it} \right] \right]$$

UPPERCASE — Indicates a COBOL reserved word. Underlined uppercase words are required. Uppercase words without underlines are optional and are used to improve readability. In either case, all uppercase words must be spelled as shown. No abbreviations are permitted.

lowercase — Indicates a generic term representing words, literals, PICTURE strings, comment entries, or a complete syntactical entry supplied by the programmer. Where *screen-name* appears in the example, enter the screen name you have chosen.

Throughout this manual, the abbreviations *id*, *id-lit*, and *lit* are used in the syntax in place of the common COBOL constructs identifier, identifier-literal, and literal.

Hyphen (-) — A hyphen appearing between uppercase words is required, as in PROGRAM-ID or SOURCE-COMPUTER. A hyphen between lowercase words indicates that your entry must not contain spaces. In the example, *screen-name* could be ACCTS-PAYABLE or ACCTSPAYABLE, but not ACCTS PAYABLE.

{ } — Braces enclosing part of a format mean that you must select one of the options enclosed within the braces. In the example, if you use the entire ACCEPT statement, you must include either COLUMN or COL.

{ } ... — A single item in braces followed by three periods indicates that you must use that item.

[ ] — Brackets enclose optional portions of a format. In the example, you have the option of specifying the line and column for the screen.

When both brackets and braces appear, they are read from the outside in. In the example, the outer brackets indicate that the entire argument is optional. If you include the argument, use only one of the elements enclosed in braces.

[ ] ...	A single item in brackets followed by three periods indicates that there can be zero or more occurrences of the item.
...	An ellipsis indicates that the item preceding it (defined by logically matching brackets or braces) can be repeated one or more times. An ellipsis between statements indicates omitted source text, commands, or other operations.
Format Punctuation	The period is required when it is present in a punctuation format. The comma and semicolon are optional, interchangeable punctuation characters used in some formats to improve readability. However, they can be used only in the positions indicated in the format. At least one space must follow a comma or semicolon used to separate statements.
Special Characters	When an arithmetic or logical operator (+, –, >, <, or =) appears in a format, it is required. These special characters are not underlined.

Contacting Data General

Data General wants to assist you in any way it can to help you use its products. Please feel free to contact the company as outlined below.

Manuals

If you require additional manuals, please use the enclosed TIPS order form (United States only) or contact your local Data General sales representative.

Telephone Assistance

If you are unable to solve a problem using any manual you received with your system, and you are within the United States or Canada, contact the Data General Service Center by calling 1–800–DG–HELPS for toll-free telephone support. The center will put you in touch with a member of Data General's telephone assistance staff who can answer your questions.

Free telephone assistance is available with your hardware warranty and with most Data General software service options. Lines are open from 8:30 a.m. to 8:30 p.m., Eastern Time, Monday through Friday.

For telephone assistance outside the United States or Canada, ask your Data General sales representative for the appropriate telephone number.

Joining Our Users Group

Please consider joining the largest independent organization of Data General users, the North American Data General Users Group (NADGUG). In addition to making valuable contacts, members receive FOCUS monthly magazine, a conference discount, access to the Software Library and Electronic Bulletin Board, an annual Member Directory, Regional and Special Interest Groups, and much more. For more information about membership in the North American Data General Users Group, call 1-800-877-4787 or 1-512-345-5316.

End of Preface

Contents

Chapter 1
Programming Concepts

Program Structure 1–1
Language Structure 1–1
- Character Sets 1–2
- Separators 1–2
- COBOL Words 1–2
- Figurative Constants 1–3
- Nonnumeric Literals 1–3
- Numeric Literals 1–3

Record Concepts 1–4
- Level Numbers 1–4
- The PICTURE Clause 1–4

Uniqueness of Reference 1–12
- Qualifying Data Division Names 1–13
- Qualifying Procedure-Names 1–13

Tables, Subscripts, Indexes 1–13
- Loading a Table 1–14
- Referencing a Table 1–16

Procedure Division Structure 1–17
- Declaratives 1–17
- Sections and Paragraphs 1–17
- Sentences and Statements 1–18
- Interactive COBOL Verbs 1–20

Arithmetic Expressions 1–20
- Arithmetic Operators 1–21
- Evaluation Order 1–21
- Overlapping Operands 1–22
- Optional Phrases 1–22

Conditional Expressions 1–23
- Simple Conditions 1–23
- Complex Conditions 1–27
- Evaluation Order of Conditions 1–29

File I/O Operations 1–30
- Current Record Pointer 1–30
- File Status Codes 1–30
- Exception Conditions 1–30

Transfer of Program Control 1–32

Chapter 2
File Organization and Access

Sequential Files 2-1
Indexed Files 2-3
The Index Portion 2-4
The Data Portion 2-4
File Efficiency 2-5
Record Deletion 2-5
Primary and Alternate Keys 2-5
File Updating 2-6
Relative Files 2-7

Chapter 3
Interactive Screen Management

Screen Data Description 3-1
Literal Format 3-3
Data-Item Format 3-4
Group Format 3-7
Screen Section Clauses 3-9
VALUE Clause 3-10
DISPLAY Clause 3-10
PICTURE Clause 3-10
FROM, TO, and USING Clauses 3-11
LINE and COLUMN Clauses 3-11
BLANK SCREEN and ERASE EOS Clauses 3-14
BLANK LINE, ERASE EOL, and ERASE LINE Clauses 3-14
BACKGROUND-COLOR and FOREGROUND-COLOR Clauses 3-15
BELL Clause 3-15
Input Control Clauses 3-15
Display Clauses 3-16
Order of Execution 3-16
Data Movement with DISPLAY and ACCEPT 3-17
Example of Interactive Screen Management 3-18

Chapter 4
COBOL Source Entry

CRT Format 4-1
Card Format 4-2
Continuation Lines 4-3
Comment Lines 4-4
Debugging Lines 4-4
Blank Lines 4-4

Chapter 5
The Identification Division

Chapter 6
The Environment Division

Configuration Section 6-2
Input-Output Section 6-3
The FILE-CONTROL Paragraph 6-3
The SELECT Clause 6-3
The I-O-CONTROL Paragraph 6-6

Chapter 7
The Data Division

The File Section 7-2
FD Entry 7-2
Working-Storage Section 7-3
Linkage Section 7-3
Screen Section 7-5
Screen Data Description Entry 7-5
Level Number 7-8
Screen-Name 7-9
Screen Clauses 7-9
Data Description Entries 7-12
Level Numbers 7-12
Data-Name/FILLER Clause 7-13
REDEFINES Clause 7-13
PICTURE Clause 7-14
USAGE Clause 7-16
SIGN Clause 7-17
OCCURS Clause 7-19
SYNCHRONIZED Clause 7-20
JUSTIFIED Clause 7-20
BLANK WHEN ZERO Clause 7-20
VALUE Clause 7-21

Chapter 8
The Procedure Division

Declarative Format 8-1
Nondeclarative Format 8-1
Terminating Procedure Division Statements 8-2
Procedure Division Statements 8-2
ACCEPT 8-3
ADD 8-9
ADD CORRESPONDING 8-10
CALL 8-12

CALL PROGRAM 8-14
CANCEL 8-17
CLOSE 8-18
COMPUTE 8-19
COPY 8-21
DELETE 8-23
DISPLAY 8-25
DIVIDE 8-27
EXIT 8-29
EXIT PROGRAM 8-30
GO TO 8-31
IF 8-33
INSPECT 8-36
MOVE 8-39
MOVE CORRESPONDING 8-41
MULTIPLY 8-43
OPEN 8-45
PERFORM 8-48
READ 8-55
REWRITE 8-61
SEARCH 8-63
SET 8-67
START 8-68
STOP 8-71
STRING 8-72
SUBTRACT 8-75
SUBTRACT CORRESPONDING 8-77
UNDELETE 8-79
UNLOCK 8-80
UNSTRING 8-81
USE 8-84
WRITE 8-86

Appendix A
ANSI Standard and Interactive COBOL Reserved Words

Appendix B
ASCII Character Sets

Glossary

Licensed Material - Property of Data General Corporation

Tables

Table

1–1 PICTURE Clause Symbols 1–5
1–2 Simple Insertion Editing 1–7
1–3 Special Insertion Editing 1–8
1–4 Fixed Insertion Editing 1–9
1–5 Floating Insertion Editing 1–10
1–6 Zero Suppression Editing 1–11
1–7 Procedure Division Verbs 1–20
1–8 Arithmetic Operators 1–21
1–9 Results with the ROUNDED Phrase 1–22
1–10 Logical Operators 1–28

2–1 File Organizations, Record Lengths, and Key Lengths 2–1
2–2 Terminators for Line Sequential Files 2–2

3–1 Screen Clauses 3–9
3–2 LINE and COLUMN Positioning 3–13

6–1 Interactive COBOL File Handling 6–5

7–1 Legal Entries in the Four Data Division Sections 7–3
7–2 Screen Section Clauses 7–11
7–3 COMPUTATIONAL Item Storage 7–17
7–4 Sign Overpunch Characters 7–18

8–1 Escape Key Codes 8–8
8–2 Screen Appearance after Execution of DISPLAY 8–26
8–3 Legal MOVE Combinations 8–40
8–4 Permissible I/O Statements with OPEN 8–46
8–5 Permissible Options with OPEN 8–47

B–1 ASCII Character Set B–1
B–2 DG International Symbols (8-bit ASCII character set) B–3

Figures

Figure

1–1 Transfer of Control Between Programs in the Run Unit 1–32

2–1 Indexed File Structure 2–4

3–1 Sample Screen Definition 3–20

4–1 COBOL Source Code in CRT Format 4–2
4–2 COBOL Source Code in CARD Format 4–3

7–1 Moving Data to Standard and Justified Fields 7–21

8–1 Flowchart for a PERFORM ... VARYING Statement 8–53
8–2 Valid PERFORM Constructs 8–54

Chapter 1
Programming Concepts

This chapter outlines the basic structure of a COBOL program, including language, data, and Procedure Division concepts.

Program Structure

A COBOL program has four divisions: Identification, Environment, Data, and Procedure. The divisions must appear in the program in the order stated.

The Identification Division provides the program name and, optionally, the program author, date of compilation, and other documentary information.

The Environment Division specifies the computers used to compile and run the program, describes special conditions, designates the logical organization of files, and describes the relationships of data files with actual input/output devices.

The Data Division describes all the data that the program uses. This includes data used in input/output operations, data developed for internal processing, and screen data.

The Procedure Division contains instructions for processing the data.

The information in a COBOL program is written in clauses (Identification, Environment, and Data divisions) and statements (Procedure Division).

A *clause* specifies an attribute of an entry. A series of clauses ending with a period is an *entry*. A *statement* specifies an action taken by the program. A series of statements ending with a period is a *sentence*.

Each clause or statement can be divided into smaller units called phrases. A *phrase* contains one or more words that are considered a syntactical unit.

Within a division, entries and sentences can be combined into larger logical units called paragraphs or sections. A *paragraph* consists of a paragraph-name followed by a period and space, and zero or more entries or sentences. In the Environment and Data divisions, paragraph names are reserved words (for example, FILE-CONTROL, I-O-CONTROL), and paragraphs contain entries. In the Procedure Division, paragraph names are programmer-defined, and contain sentences.

In the Environment and Data divisions, a *section* consists of a section header and zero or more entries. The section header contains a reserved section name followed by the word SECTION. A section header must be followed by a period and a space. In the Procedure Division, a section consists of a programmer-defined section name, followed by a period and space, and zero or more paragraphs.

Language Structure

COBOL is an English-like language using sentences composed of characters, words, and statements. Interactive COBOL includes the language components described in the following sections.

Character Sets

Interactive COBOL uses the following three character sets:

- The COBOL character set. This is a set of the 51 characters that are legal in COBOL. The characters include the uppercase letters A–Z, the digits 0–9, space, and the following special characters: + – * / = > < $, . " ( ) ;
In addition, revision 1.60 or greater of Interactive COBOL also accepts lowercase letters a–z. Within a COBOL program, the lowercase letters are equivalent to the uppercase letters, except in nonnumeric literals.
- The 96-character ASCII set. Characters in this set are used to form data-items and nonnumeric literals.
- The DG international character set. This set, which can be used on terminals equipped to handle 8-bit ASCII values, includes 69 displayable characters in addition to the 96-character ASCII set.

Separators

In a COBOL source program, separators delimit words and sentences. The separators are:

- Space character. When a format indicates a space, more than one can be used.
- Tab. A Tab is equivalent to a space.
- Comma, semicolon, and period. A space must follow these characters.
- Left and right parentheses. Use these in balanced pairs to denote subscripts, indexes, arithmetic expressions, or conditions.
- Quotation marks. Use these in balanced pairs to delimit nonnumeric literals. An opening quotation mark must be preceded by a space or a left parenthesis; a closing quotation mark must be followed by one of the following separators: space, comma, semicolon, period, or right parenthesis.

COBOL Words

Words can be up to 30 characters long. There are two classes of COBOL words: programmer-defined words and reserved words.

Programmer-defined words are supplied by the programmer to satisfy the requirements of a specific format. A programmer-defined word consists of the characters a–z (revision 1.60 or greater of Interactive COBOL only), A–Z, 0–9, and hyphen. It must not contain spaces, and the hyphen cannot be the first or last character. Level numbers and segment numbers must be numeric. Programmer-defined words, except for paragraph and section names, must contain at least one alphabetic character.

All programmer-defined words must be unique within a program. A word is unique if no other word has the identical spelling or punctuation, or if uniqueness can be ensured by qualification (see "Uniqueness of Reference" later in this chapter).

Reserved words are assigned a special meaning within the COBOL language; they can be used only as indicated in the syntax formats. Appendix A contains a list of ANSI Standard and Interactive COBOL reserved words.

Figurative Constants

COBOL has several reserved words that represent specific constant values. These figurative constants are used in a program to avoid repetitive coding. The singular and plural forms are interchangeable. Figurative constants must not be enclosed in quotation marks. The figurative constants and their values are:

ZERO, ZEROS, ZEROES	The digit 0
SPACE, SPACES	The space character
HIGH-VALUE, HIGH-VALUES	Octal value 377
LOW-VALUE, LOW-VALUES	Octal value 000
QUOTE, QUOTES	The quotation mark character
ALL literal	The specified literal. The literal must be either a nonnumeric literal or a figurative constant. However, the ALL figurative-constant form is redundant.

If you move a figurative constant to a receiving field, the receiving field is filled by the value of the figurative constant. For example, the statement MOVE ZERO TO A-FIELD fills A-FIELD with zeroes.

If you define A-FIELD as a seven character field, the statement MOVE ALL "ABCD" TO A-FIELD fills A-FIELD with the value "ABCDABC".

Nonnumeric Literals

A nonnumeric literal is a character string containing elements from the ASCII character set. The literal must be enclosed in quotation marks. Its size can be up to 160 characters (excluding the quotation marks) for revision 1.60 or greater of Interactive COBOL and 132 characters (excluding the quotation marks) for all other revisions of Interactive COBOL. The value of the literal is the string of characters between the quotation marks. Embedded spaces are counted as part of the literal. Quotation marks within a literal are represented by two consecutive quotation marks. For example:

```
MOVE "THE DATE IS" TO HEAD-LINE.
DISPLAY "INPUT ERROR, PLEASE RE-ENTER THE DATA!".
MOVE "-The current time is:" TO TIME-LINE.
DISPLAY "The title is ""Learning to Program""."".
```

Numeric Literals

A numeric literal is a character string selected from the set 0–9, plus, minus, and decimal point. It can contain up to 18 digits. An unsigned literal is positive. If a sign is used, it must be the leftmost character. A numeric literal cannot contain more than one sign character. If a decimal point is used, it cannot be the rightmost character.

The following are legal numeric literals:

```
ADD 122 TO CATEGORY.
MULTIPLY 3.1416 BY DIAM GIVING CIRCUM.
MOVE -167 TO TEMP.
```

The following are illegal numeric literals:

```
ADD 1543. TO CATEGORY. (Decimal is on the right.)
MULTIPLY 3.14+ BY DIAM GIVING CIRCUM. (Sign is not on the left.)
MOVE 1234567890987654321 TO TEMP. (Literal exceeds 18 digits.)
```

Record Concepts

A COBOL program deals with data grouped into files. Within a file, the data is logically organized by records. A COBOL *record* consists of uniquely identifiable items treated as a unit. A record is the most inclusive COBOL data-item; it can be divided into logical subdivisions, which may in turn be further subdivided. A record or record division that has no subdivisions is called an *elementary item*. An item that is subdivided is called a *group item*, and the entire record structure with all its subdivisions is referred to as a *hierarchy*. The programmer must assign a level number and a unique name to each elementary and group item within the hierarchy.

Level Numbers

Level numbers in a data description show the organization of elementary and group items, and identify special-purpose data-items. Because records are the most inclusive data-items, they are assigned 01 level numbers. Subdivisions of the record are given successively higher level numbers. The level numbers 02 through 49 identify elementary and group items within a record.

There are two special level numbers: 77 and 88. Level 77 identifies an independent elementary item in the Working-Storage or Linkage sections. An independent item is not in a hierarchy, and is not part of a record. Level 77 items describe noncontiguous data. Level 88 gives condition-names to values contained in elementary data-items.

The PICTURE Clause

The PICTURE clause occurs in the Data Division of a COBOL program. It specifies the size and data type of an elementary data-item. Every elementary data-item except those described as USAGE IS INDEX must have a PICTURE clause.

The format of the PICTURE clause is:

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

The words PICTURE and PIC are synonymous. The number of characters in the picture-string determines the size of the data-item, and the specific characters in the picture-string determine the type of data-item. A data-item can be alphabetic, numeric, alphanumeric, numeric edited, or alphanumeric edited. Picture strings can contain uppercase or lowercase letters in Interactive COBOL revision 1.60 or greater.

The following sections explain data types and how to specify them in the PICTURE clause.Table 1-1 lists the PICTURE clause symbols and their meanings.

Alphabetic Data-Items

The symbol A describes alphabetic data-items. For example:

```
05  STATE-ABBREV  PIC AA.
```

The data-item STATE-ABBREV is alphabetic and two characters long.

Table 1-1 PICTURE Clause Symbols

Symbol	Meaning
X	Alphanumeric character
A	Alphabetic character or space
9	Numeric character
V	Assumed decimal point
S	Arithmetic sign
P	Assumed decimal scaling position
B	Space insertion
/	Slash insertion
0	Zero insertion
.	Decimal point insertion
,	Comma insertion
+	Plus sign insertion
-	Minus sign (if negative value) or blank insertion
CR	CR (credit sign) insertion
DB	DB (debit sign) insertion
$	Currency symbol insertion
Z	Zero suppression by space
*	Zero suppression by asterisk

Numeric Data-Items

The symbols 9, V, S, and P describe numeric data-items. A numeric data-item contains the characters 0 through 9. If you include the S symbol, the item contains a plus or a minus sign. Data-items used in arithmetic computations must be described as numeric.

A numeric data-item can be 1 to 18 characters long, 19 if the SIGN IS SEPARATE clause is present.

Each numeric picture-string must contain at least one 9. The 9 represents a numeric digit position containing a value from 0 to 9.

The V symbol represents an assumed decimal point. It specifies a location but not an actual storage position. It is not counted in the size of a data-item. A picture-string can contain only one V; if it is the rightmost character, it is redundant.

The S symbol indicates a signed data-item, and is not counted in the size of a data-item. Only one S is allowed in a picture-string, and it must be the leftmost character. You must include the S symbol if you specify the SIGN IS SEPARATE clause.

The P symbol indicates an assumed decimal scaling position when there is no decimal point in the data-item. The P specifies leading or trailing digits and scales the number by powers of 10. The data-item is assumed to contain a zero in each position held by a P, but no zeros are actually stored.

The P is not counted in the size of the data-item. However, it is counted when determining the number of digit positions in numeric and numeric-edited items. Thus if the data-item evaluates to a number exceeding 18 digits, you cannot perform arithmetic operations with it.

P can appear only in a sequence in the rightmost or leftmost portion of the format. The implicit decimal point is assumed to be on the right if the string of Ps is to the right; the decimal point is assumed to be on the left if the Ps are to the left. PICTURE 999P(6) represents a number in the millions, storing only the three leftmost digits. PICTURE PPP999 represents a decimal fraction in the millionths, storing only the three rightmost digits.

If a data-item has a P in its picture-string, any value moved to the data-item must reflect the scaled form. For example, if AMOUNT-IN is defined as PIC 999PPP and contains the value 238, and AMOUNT-OUT is defined as PIC 9(6), the statement MOVE AMOUNT-IN TO AMOUNT-OUT will cause AMOUNT-OUT to contain the value 238000.

If a data-item has a VALUE clause, the value specified must be in the scaled form (for example, PIC 999PPP VALUE 238).

If you display a scaled data-item, the unscaled version appears on your screen.

The following examples show numeric data-items:

```
15  ACCT-NUM     PIC 9(10).
```

The data-item ACCT-NUM is unsigned numeric with 10 digits.

```
10  COST      PIC 99999V99.
```

The data-item COST is unsigned numeric with five digits to the left of the assumed decimal point and two digits to the right. The total number of characters is seven; the V is not counted in the total.

```
10  TOTAL-PAY    PIC S9(5)V99.
```

The data-item TOTAL-PAY is signed numeric with five digits to the left of the assumed decimal point and two digits to the right. The total number of characters is seven; neither the V nor the S (unless the SIGN IS SEPARATE clause is present) is counted in the total.

```
10  MEGA-VALUE        PIC 9999PPPPPP.
```

The data-item MEGA-VALUE has four digits and six scaling positions. If the value stored is 241, the effective, or scaled, value is 241,000,000.

```
10  CENTI-VAL         PIC PP99999.
```

The data-item CENTI-VAL has five digits and two scaling positions. If the value stored is 1000, the effective, or scaled, value is .0001.

Alphanumeric Data-Items

The symbols A, X, and 9 describe alphanumeric data-items. The picture-string of an alphanumeric data-item must conform to one of the following rules:

- It must contain at least one A and at least one 9.
- It must contain at least one X.

An alphanumeric data-item is treated as though its picture-string contains all Xs.

The following examples describe alphanumeric data-items:

```
10  SS-NUM       PIC X(11).
```

The data-item SS-NUM is alphanumeric and 11 characters long.

```
10  MIXED-UP-PIC     PIC AA99.
```

The data-item MIXED-UP-PIC is alphanumeric and four characters long. The picture-string is equivalent to PIC X(4).

Edited Data-Items

A COBOL program can edit numeric, alphanumeric, and alphabetic data-items. There are five types of editing: simple insertion, special insertion, fixed insertion, floating insertion, and zero suppression. You can use simple insertion editing on both numeric and alphanumeric data-items; the other types of editing are valid only on numeric data-items.

Simple Insertion

You can edit numeric and alphanumeric data-items with simple insertion editing. The insertion characters are comma (numeric data-items only), B (space), 0, and /. These characters represent the position in the item where the character will be inserted. Insertion characters are counted in the length of an item. Table 1-2 gives examples of simple insertion editing.

Table 1-2 Simple Insertion Editing

PICTURE	Data Value	Result
XXXBXXBX(4)	FEB111990	FEB 11 1990
99,999	36895	36,895
A(5)BA(5)	ABCDEFGHIJ	ABCDE FGHIJ
XX/XX/XX	121348	12/13/48
99,000,000	17	17,000,000

Special Insertion

The decimal point is the special insertion symbol; it controls data-item alignment. You cannot use an actual decimal point and the implied decimal point (V) in the same picture-string. When you move data from an item having an assumed decimal point to

an item having an actual decimal point, the data is placed in the receiving item with the actual decimal point aligned with the implied decimal point (see Table 1-3). The decimal point cannot be the last character in the character string.

If you use DECIMAL-POINT IS COMMA in the SPECIAL-NAMES paragraph, the rules applying to periods and commas in the PICTURE clause are reversed.

Table 1-3 Special Insertion Editing

Sending PIC	Receiving PIC	Data Value	Result
9(5)	999.99	12345	345.00
999V99	999.99	12345	123.45
99V999	999.99	12345	012.34
9(4)V9	999.99	12345	234.50

Fixed Insertion

Fixed insertion editing uses the following five symbols : +, –, CR, DB, and the currency symbol (usually a $). A picture-string can contain only one currency symbol and only one other editing symbol (+, –, CR, or DB).

The + or – must occupy either the leftmost or the rightmost position. If you specify the + in the picture-string, Interactive COBOL inserts a + character in the data-item if the value stored is positive or zero, and inserts a – if the value is negative. If you specify the –, Interactive COBOL inserts a space in the data-item if the value is positive or zero, and inserts a – if the value is negative.

The symbols CR and DB count as two character positions in determining the size of the item. They must always appear in the rightmost position. If you specify CR in the picture-string, Interactive COBOL inserts two spaces after the data-item if the value is positive or zero, and inserts CR if the value is negative. If you specify DB, Interactive COBOL inserts two spaces after the data-item if the value is positive or zero, and inserts DB if the value is negative.

The currency symbol must occupy the leftmost position, except that either a + or a – can precede it. The currency symbol is usually the dollar sign ($). However, if you specified the CURRENCY SIGN IS clause, use the symbol defined in that clause.

See Table 1–4 for examples of fixed insertion editing.

Table 1-4 Fixed Insertion Editing

PICTURE	Data Value	Result
$9999.99	+1234.50	$1234.50
	-1234.50	$1234.50
	0	$0000.00
9999.99CR	+1234.50	1234.50
	-1234.50	1234.50CR
	0	0000.00
9999.99DB	+1234.50	1234.50
	-1234.50	1234.50DB
	0	0000.00
+9999.99	+1234.50	+1234.50
	- 1234.50	-1234.50
	0	+0000.00
9999.99+	+1234.50	1234.50+
	-1234.50	1234.50-
	0	0000.00+
-9999.99	+1234.50	□1234.50
	-1234.50	-1234.50
	0	□0000.00
9999.99-	+1234.50	1234.50
	-1234.50	1234.50-
	0	0000.00

Floating Insertion

Three of the fixed insertion symbols also serve as floating insertion symbols: +, -, and the currency symbol. Floating insertion editing suppresses leading zeros and places the editing symbol immediately before the first significant digit. The floating symbols are mutually exclusive, except that a plus or minus used as a fixed insertion character can appear to the left of a floating currency symbol.

Floating insertion occurs when you specify one of the floating insertion symbols two or more times within the picture-string. However, the symbol appears only once in the output. The string of floating insertion characters can contain the B, zero, comma, slash, or decimal point. The characters are considered part of the floating string.

There are two ways to represent floating insertion editing in a picture-string:

1. Represent any or all leading numeric character positions to the left of the decimal point by the insertion character (for example, $$$9.99 or $$$.99).

2. Represent all numeric character positions by the insertion character (for example, $$$.$$).

For example, if you specify PIC $,$$$,999.99 for a data-item and its value is 12345, the edited result is □□□□□$123.45. Note that both commas are suppressed in the edited result.

The leftmost character of the floating string represents the leftmost position of the floating symbol in the data-item. The rightmost character of the floating string represents the rightmost position.

The second leftmost floating insertion symbol represents the leftmost limit at which numeric data can appear in the data-item. Nonzero numeric data can replace all characters at or to the right of this limit. For example, with PIC $$$$$.99, a maximum of four digits can appear to the left of the decimal point (for example, $1234.00).

When floating insertion characters appear only to the left of the decimal point, the following occurs: a single floating insertion character is placed immediately to the left of the first nonfloating position, the first nonzero digit in the data, or the decimal point. Positions to the left of the inserted character fill with spaces (see Table 1-5 for examples).

If all character positions are represented by a floating insertion symbol and the data value is zero, the entire data-item contains spaces. If the value is not zero, the result is the same as if the insertion character appeared only to the left of the decimal point in the picture-string.

Table 1-5 Floating Insertion Editing

PICTURE	Data Value	Result
$$9.99	.23	□$0.23
$$$.99	1.23	□$1.23
$$$.99	.23	□□$.23
$$$9.99	.12	□□$0.12
$$$$,$$$.99	000123	□□□□$123.00
-------.00	123456	□123456.00
$$,$$$,$$$.99CR	-1234567	$1,234,567.00CR
++,+++,+++	0	□□□□□□□□□□
++,+++	123	□□+123

Zero Suppression

The symbols Z and asterisk (*) control the suppression of leading zeros in a data-item. With Z the replacement character is the space; with an asterisk the replacement character is an asterisk. The Z and * cannot appear together in a picture-string.

Any of the simple insertion characters (comma, B, 0, and /) can appear within or immediately to the right of the suppression string. These characters are counted in the length of the string.

Suppression symbols can represent all digit positions in the data-item (for example, ZZZ.ZZ), or any leading digit positions to the left of the decimal point (for example, ZZZ9.99).

If suppression symbols appear only to the left of the decimal point, any leading zero in the data that corresponds to a suppression symbol is replaced by a space or

asterisk. Suppression terminates at the first nonzero digit or at the decimal point, whichever is encountered first.

If all numeric 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 appeared only to the left of the decimal point. If the value is zero and the suppression symbol is Z, the entire data-item contains spaces. If the value is zero and the suppression symbol is asterisk, the data-item contains all asterisks, except for the actual decimal point.

A PICTURE clause using the asterisk as the zero suppression symbol cannot appear in the same entry with the BLANK WHEN ZERO clause. Table 1-6 gives examples of zero suppression editing.

Table 1-6 Zero Suppression Editing

PICTURE	Data Value	Result
Z,ZZZ.99	1234.56	1,234.56
ZZZZ.ZZ	1234	1234.00
$Z,ZZZ.99	123.45	$□□123.45
ZZZZZ.ZZ	.01	□□□□□.01
*****.**	0	*****.**
*****.99	0	*****.00
ZZZZZ.ZZ	0	□□□□□□□□

Alignment of Data-Items

The standard rules for positioning data within an elementary item depend on the category of the receiving item. The alignment rules for numeric receiving items are:

- If you do not specify a decimal point, assume it exists immediately to the right of the field.
- The data aligns on the decimal point and, if necessary, the value is truncated or padded with zeros at either end.

The alignment rule for numeric edited receiving items is:

- The data aligns on the decimal point and, if necessary, the value is truncated or padded with zeros at either end, except where editing causes replacement of leading zeros.

The alignment rules for alphabetic, alphanumeric, and alphanumeric edited receiving items are:

- The data aligns at the leftmost character position and, if necessary, the value is truncated or padded with spaces to the right.
- If you specify the JUSTIFIED clause for a receiving item, the above rule is modified as described in the JUSTIFIED clause.

Uniqueness of Reference

Programmer-defined names must be unique, either through different spellings or through qualification. Duplicate names can appear in separate hierarchies, but the names must be made unique by references to higher-level names, or *qualifiers,* within each hierarchy. Data Division names and paragraph-names can be qualified. Filenames, record-names, section-names, index-names, level 77 Working-Storage data-names, and level 77 Linkage Section data-names cannot be qualified.

Data-name qualifiers must be separated by OF or IN. OF and IN are interchangeable; use the term that is most readable. The format for a data-name qualifier is:

$$\textit{identifier} \left[\left\{ \begin{matrix} \underline{\text{OF}} \\ \underline{\text{IN}} \end{matrix} \right\} \textit{identifier} \right] \ldots$$

Specify qualifiers in ascending order within the hierarchy until the data-name is uniquely qualified. Use as many qualifiers as necessary to make the name unique.

Consider the following example:

```
WORKING-STORAGE SECTION.
01  COLOR       PIC X(5).
01  PART-DESCRIPTION.
    03 COLOR    PIC X(5).
```

If the program makes no unqualified reference to COLOR, the above code will not cause a compiler error. Thus, the following statement is valid and the compiler does not issue a duplicate name warning:

```
MOVE "RED" TO COLOR OF PART-DESCRIPTION.
```

If the program does make an unqualified reference to COLOR, an error occurs and the compiler generates a message stating the data-name is ambiguous. Thus, the following statement is invalid:

```
MOVE "RED" TO COLOR.
```

In the above Working-Storage example, the 01 identifier COLOR can never be used in the program because it is ambiguous. However, the fully qualified COLOR OF PART-DESCRIPTION is not ambiguous and can be used.

Licensed Material - Property of Data General Corporation

Qualifying Data Division Names

In Data Division references, all data-name qualifiers must be associated with a filename or level number. In the hierarchy of qualification, a filename is the most significant qualifier. The names associated with level number 01 are the next most significant, followed by the names associated with level numbers 02, 03, and so on. For a data-item in the File Section, the highest qualification is a filename. For a data-item in the Working-Storage Section, the highest qualification is an 01 item. The following example illustrates qualified references to duplicate data-names:

```
DATA DIVISION.
...
FD  I-TAX-FILE...                FD  O-TAX-FILE...
01  SINGLE-REC...                 01  TAX-REC...
    03 EMPLOYEE-NAME...                03 EMPLOYEE-NAME...
       05 FIRST-NAME...                   05 FIRST-NAME...
       05 LAST-NAME...                    05 LAST-NAME...
    03 ADDR...                         03 ADDR...
    03 GROS...                         03 GROS...
    03 NET...                          03 NET...
    03 TAX...                          03 TAX...

PROCEDURE DIVISION.
...
MOVE LAST-NAME IN EMPLOYEE-NAME IN SINGLE-REC TO
    LAST-NAME IN EMPLOYEE-NAME IN TAX-REC.
MULTIPLY GROS OF I-TAX-FILE BY .024 GIVING
    TAX IN TAX-REC.
```

Qualifying Procedure-Names

Two or more paragraphs in the Procedure Division can have identical names if they appear in different sections. A paragraph-name can be qualified only by a section name. The keyword SECTION cannot appear in the entry. An example of paragraph-name qualification is:

```
PERFORM PARAGRAPH-1 OF QUALIFYING-SEC.
```

A paragraph-name need not be qualified when referred to within the section in which it appears.

Tables, Subscripts, Indexes

A table is a set of contiguous data-items with one name and one data description, and is defined by the OCCURS clause. Subscripts and indexes refer to elements in a table. A subscripted data-item has the format:

data-name (*subscript-1* [, *subscript-2* [, *subscript-3*]])

The subscript is an integer or an identifier that evaluates to an integer.

An indexed data-item has the format:

data-name (*index-1* [, *index-2* [, *index-3*]])

The index can have any of the following formats: identifier, identifier + integer, or identifier – integer.

Interactive COBOL treats an index in much the same way as a subscript. For both subscripts and indexes:

- The data-name must be the name of a table element, and it can be qualified.
- Parentheses immediately follow the data-name. One or more spaces can precede the opening parenthesis.
- The lowest possible subscript or index value is 1; the highest possible value is the maximum number of elements in the table. The OCCURS clause specifies this value.
- Three levels of subscripts and indexes are allowed.

Identifier subscripts are defined in the File, Working-Storage, and Linkage sections. Change a subscript value the same way as any other data-item—move values to it, add values to it, or subtract values from it.

Declare indexes in the Working-Storage Section with the INDEXED BY phrase of the OCCURS clause. The SET and PERFORM verbs change index values, which allows the data-name to point to different items in the table.

Loading a Table

Your COBOL program must insert values into, or load, the table. There are two ways to load a table: hard-coding values into the program, and entering table values at execution time either interactively or from a file.

Use the VALUE clause to hard-code data values into a table. The following example shows a subscript and a table that includes salary grades and the corresponding hourly pay.

```
WORKING-STORAGE SECTION.
...
77  PAY-SUB         PIC 99.

01  PAY-VALUES.
    05  FILLER      PIC X(6)     VALUE "020500".
    05  FILLER      PIC X(6)     VALUE "040550".
    05  FILLER      PIC X(6)     VALUE "060600".
    05  FILLER      PIC X(6)     VALUE "080650".
    05  FILLER      PIC X(6)     VALUE "100700".

01  PAY-TABLE REDEFINES PAY-VALUES.
    05  PAY-TABLE-ENTRY OCCURS 5 TIMES.
        10  GRADE-ITEM      PIC 99.
        10  RATE-ITEM       PIC 99V99.
```

Use the same format for indexes, except:

- Do not define a subscript (PAY-SUB, in this example).
- Include the INDEXED BY phrase of the OCCURS clause. In this example, the coding could be:

```
05  PAY-TABLE-ENTRY OCCURS 5 TIMES INDEXED BY ITEM-IND.
```

A data-item cannot have both a VALUE clause and an OCCURS clause.

If a table will contain volatile data, you can enter it during program execution. There are two ways to do this. The Interactive COBOL program can read the table items from a data file, or you can enter the table items interactively.

To set up a program that reads table items from a data file, define the data file in your program with the SELECT clause and an FD entry, and process it with OPEN, READ, and CLOSE statements. Since you will not insert the table item values with the VALUE clause, you do not need to redefine the table.

Using the same pay grade and rate table as above, define and load the table as follows:

```
FILE-CONTROL.
    SELECT PAY-FILE ASSIGN TO DISK, "PAY$FILE".
FILE SECTION.

FD  PAY-FILE
    RECORD CONTAINS 6 CHARACTERS
    LABEL RECORDS ARE STANDARD.
01  PAY-RECORD.
    05  GRADE        PIC 99.
    05  RATE         PIC 99V99.
...
WORKING-STORAGE SECTION.
...
77  SUB              PIC 99.
...
01  PAY-TABLE.
    05  PAY-TABLE-ENTRY OCCURS 5 TIMES.
        10  GRADE-ITEM      PIC 99.
        10  RATE-ITEM       PIC 99V99.

PROCEDURE DIVISION.
...
    PERFORM LOAD-TABLE VARYING SUB FROM 1 BY 1
        UNTIL SUB > 5.

LOAD-TABLE.
    READ PAY-FILE.
    MOVE GRADE TO GRADE-ITEM (SUB).
    MOVE RATE TO RATE-ITEM (SUB).
```

In this example, the disk file PAY$FILE contains the data for the table.

You can also load the table interactively, using the Interactive COBOL screen management facility discussed in detail in Chapter 3. The following example shows how to load the table interactively:

```
WORKING-STORAGE SECTION.
01  PAY-TABLE.
    05  PAY-TABLE-ENTRY OCCURS 5 TIMES
            INDEXED BY ITEM-IND.
        10  GRADE-ITEM      PIC 99.
        10  RATE-ITEM       PIC 99V99.

SCREEN SECTION.
01  PAY-SCREEN.
    05  BLANK SCREEN.
    05  LINE 2 COL 20 VALUE "EMPLOYEE PAY GRADES AND RATES".
    05  LINE 5 COL 10 VALUE "Pay Grades".
    05  LINE 5 COL PLUS 10 VALUE "Pay Rates".
    05  LINE PLUS 2 COL 13 PIC 99 TO GRADE-ITEM (ITEM-IND).
    05  LINE 7 COL PLUS 10 PIC 99.99 TO RATE-ITEM (ITEM-IND).
    05  LINE PLUS 1 COL 13 PIC 99 TO GRADE-ITEM (ITEM-IND + 1).
    05  LINE 8 COL PLUS 10 PIC 99.99 TO RATE-ITEM (ITEM-IND + 1).
    05  LINE PLUS 1 COL 13 PIC 99 TO GRADE-ITEM (ITEM-IND + 2).
    05  LINE 9 COL PLUS 10 PIC 99.99 TO RATE-ITEM (ITEM-IND + 2).
    05  LINE PLUS 1 COL 13 PIC 99 TO GRADE-ITEM (ITEM-IND + 3).
    05  LINE 10 COL PLUS 10 PIC 99.99 TO RATE-ITEM (ITEM-IND + 3).
    05  LINE PLUS 1 COL 13 PIC 99 TO GRADE-ITEM (ITEM-IND + 4).
    05  LINE 11 COL PLUS 10 PIC 99.99 TO RATE-ITEM (ITEM-IND + 4).

PROCEDURE DIVISION.
    ...
    SET ITEM-IND TO 1.
    DISPLAY PAY-SCREEN.
    ACCEPT PAY-SCREEN.
```

Referencing a Table

Tables are referred to in the Procedure Division. To refer to a specific item in a table, specify its data-name and subscript (or index). For two- or three-dimensional tables, specify a subscript for each dimension. The compiler associates the rightmost subscript in a list with the innermost dimension of the array. Therefore, list subscript levels in order of successively smaller inclusions and separate them with spaces or other separators.

You refer to a table in different ways, depending on whether it is subscripted with an integer or data-item, or whether it is indexed. In the following examples, the third value in PAY-TABLE is moved to the data-name TEMP.

```
MOVE GRADE-ITEM (3) TO TEMP.
```

```
MOVE 3 TO SUB.
MOVE GRADE-ITEM (SUB) TO TEMP.

SET ITEM-IND TO 3.
MOVE GRADE-ITEM (ITEM-IND) TO TEMP.
```

You can also change subscript and index values by coding table-manipulation statements in a PERFORM VARYING loop, varying the subscript or index as required.

Procedure Division Structure

The Procedure Division is the last division in a COBOL program. It contains the logic of the program. The program logic is written in statements that begin with a COBOL verb, which expresses an action taken by the program. The words and symbols that make up the rest of the statement specify what the COBOL verb is to act upon.

Chapter 8 lists the format of the Procedure Division. The Procedure Division begins with the header, which, like other division headers, is a required entry. Under the header, in decreasing order, are the Procedure Division declaratives, sections, paragraphs, sentences and statements, and verbs.

Declaratives

The Declaratives section, if used, must appear at the beginning of the Procedure Division. The section must be preceded by the reserved word DECLARATIVES and must end with the reserved words END DECLARATIVES. The declaratives section contains procedures to handle exceptional I/O conditions that arise during execution of the file handling statements CLOSE, DELETE, OPEN, READ, REWRITE, START, UNDELETE, UNLOCK, and WRITE. Each section in DECLARATIVES contains a USE statement identifying the conditions under which the associated procedures are executed.

Procedures specified in the Declaratives section are executed under either of the following conditions:

- When the AT END or INVALID KEY condition exists and you do not specify the appropriate phrase in the I/O statement
- When some other I/O error occurs

After execution of a USE procedure, control returns to the statement following the statement that caused the error.

See the "Declaratives Format" and the USE statement in Chapter 8 for more information.

Sections and Paragraphs

A COBOL program is structured with sections and paragraphs or with paragraphs only. The format for a section is:

section-name SECTION [*segment-number*].
{ *paragraph-name*. [*sentence*]...}...

The format for a paragraph is:

paragraph-name. [*sentence*]...

The above examples indicate that the Procedure Division can contain, at the minimum, either one section-name, the word SECTION,and one paragraph-name, or one paragraph-name. Paragraph-name is required if there is a section-name. Although this would be a valid syntactical construction, it would perform nothing. Sentences and statements perform the logic of the program. Statements must be contained in sentences, and sentences, in turn, must be contained in paragraphs. Sections are optional only if there is no Declaratives section. If there is a Declaratives section, section-name must be present after the END DECLARATIVES statement.

The out–of–line PERFORM and GO TO statements refer to sections and paragraphs. Segment-number is used for documentation only.

Sentences and Statements

A sentence is a statement or series of statements followed by a period and one or more spaces. Although this difference may seem trivial, it has a profound effect on your program logic.

For example, the following code is one sentence. Both COMPUTE statements are executed if BYPASS-REC-SW has a value of N. However, if you add a period to the end of the second line, thus making two sentences, the third line becomes a separate sentence and is performed unconditionally.

```
IF BYPASS-REC-SW = "N"
    COMPUTE NET = GROSS - TOT-TAXES
    COMPUTE YTD-NET = NET + OLD-NET.
```

In the following example, the code in SCREEN-PARA is performed unconditionally. Lines 2 and 3 can be sentences (periods) or statements (no periods). The only constraint is that the last statement in the paragraph end with a period.

```
SCREEN-PARA.
    SET ITEM-INDEX TO 1.
    DISPLAY PAY-SCREEN.
    ACCEPT PAY-SCREEN.
```

The computer executes each sentence sequentially until it encounters a branching sentence, such as one containing a GO TO or CALL statement.

The next example combines several statements into one sentence. When A-RECORD is SPACES, three statements are executed: the IF, the READ, and the DISPLAY statements. These statements may cause the statements within them to be executed: the MOVE and the PERFORM statements. The period after COUNT-SCREEN ends the sentence.

```
IF A-RECORD = SPACES
    IF B-RECORD-FLAG = "Y"
        MOVE "N" TO B-RECORD-FLAG
        MOVE 1 TO USE-COUNT
```

```
END-IF

READ A-FILE RECORD
    AT END PERFORM EOF-ROUTINE
            MOVE ZERO TO USE-COUNT
END-READ

DISPLAY COUNT-SCREEN.
```

Conditional Statements

A conditional statement is evaluated at runtime, and your program takes an action based on whether the expression is true or false. Interactive COBOL allows the following conditional statements:

- IF statement
- READ statement with an AT END or INVALID KEY phrase
- WRITE, START, REWRITE, or DELETE statement with an INVALID KEY phrase
- An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) with a SIZE ERROR phrase
- CALL statement with an ON EXCEPTION or ON OVERFLOW phrase
- CALL PROGRAM statement with an ON EXCEPTION phrase
- ACCEPT statement with an ON ESCAPE phrase

If you use an explicit scope terminator, such as END–CALL, an otherwise conditional statement becomes an imperative statement.

Imperative Statements

An imperative statement indicates a specific, unconditional program action. An imperative statement can consist of a sequence of imperative statements or it may be a conditional statement that contains an explicit scope terminator, such as verb with the prefix END– (see Chapter 8 for a list of these verbs). Any statement that is not a conditional statement or a compiler directing statement is an imperative statement. Compiler directing statements begin with the COPY or USE verb and direct the compiler to perform specific actions.

You can use the verbs listed in Table 1–7 in imperative statements. However, unless you include its explicit scope terminator (for example, END–ADD) the statement becomes conditional if you use any of the following optional phrases with a verb: SIZE ERROR, INVALID KEY, ON EXCEPTION, ON OVERFLOW, AT END, ON ESCAPE.

The following example is illegal because ADD C TO D is a conditional statement:

```
ADD A TO B
    ON SIZE ERROR
        PERFORM CHECK
```

```
ADD C TO D
    ON SIZE ERROR
        PERFORM EXIT-RTN.
```

The example becomes legal when you include the explicit scope terminator for ADD:

```
ADD A TO B
    ON SIZE ERROR
        PERFORM CHECK
        ADD C TO D
            ON SIZE ERROR
                PERFORM EXIT-RTN
        END-ADD.
```

Interactive COBOL Verbs

Table 1-7 lists the categories of Interactive COBOL verbs. Each of these verbs is discussed in detail in Chapter 8.

Table 1-7 Procedure Division Verbs

Arithmetic Operation	Data Manipulation
ADD	INSPECT
ADD CORRESPONDING	MOVE
COMPUTE	MOVE CORRESPONDING
DIVIDE	SET
MULTIPLY	SEARCH
SUBTRACT	STRING
SUBTRACT CORRESPONDING	UNSTRING
	Compiler Directing
	COPY
Input/Output	USE
ACCEPT	**Transfer of Control**
CLOSE	
DELETE	CALL
DISPLAY	CALL PROGRAM
OPEN	CANCEL
READ	EXIT
REWRITE	EXIT PROGRAM
START	GO TO
UNDELETE	IF
UNLOCK	PERFORM
WRITE	STOP

Arithmetic Expressions

The verbs used to form arithmetic statements are ADD, DIVIDE, MULTIPLY, SUBTRACT, and COMPUTE. They have several common features:

- PICTURE clauses of data-items must be numeric (9, S, V, P). However, they need not be identical. Any necessary conversion and decimal point alignment is done

throughout the calculation. Data-items that you use only to store results in can be numeric-edited, for example, in the phrase ADD A TO B GIVING C, the data-item C can be numeric-edited.

- The maximum size of each expression is 18 digits.
- The composite of operands must not exceed 18 digits. The composite is a hypothetical data-item resulting from superimposing all expressions in a statement, aligned on their decimal points.

Two optional phrases can appear in all arithmetic statements: the ROUNDED phrase and the SIZE ERROR phrase. These phrases, described later in this chapter, allow more specific control over computed results.

Arithmetic Operators

Five binary and two unary arithmetic operators can be used in arithmetic expressions (see Table 1-8). A space must precede and follow these operators.

Table 1-8 Arithmetic Operators

Symbol	Meaning
Binary	
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
Unary	
+	The effect of multiplication by +1.
-	The effect of multiplication by -1.

Evaluation Order

An arithmetic expression is a combination of identifiers, literals, and arithmetic operators that evaluates to a single numeric value. The evaluation order within an expression is:

1. Unary plus and minus
2. Exponentiation in left-to-right order
3. Multiplication and division in left-to-right order
4. Addition and subtraction in left-to-right order

Parentheses can change the order of evaluation. Elements within parentheses are evaluated first. With nested parentheses, the innermost expression is evaluated first. Each left parenthesis must have a corresponding right parenthesis.

Overlapping Operands

When a sending and a receiving item in an arithmetic statement or an INSPECT, MOVE, or SET statement share a part of their storage areas, the result of the execution of the statement is undefined. The following example illustrates an *incorrect* move:

```
01  A.
    02  B.
        03  C PIC X.
        03  D.
            04  E PIC X.
            04  F PIC X.

    MOVE B TO D.
```

Optional Phrases

Interactive COBOL provides two optional phrases: the ROUNDED phrase and the SIZE ERROR phrase. These phrases are legal in all arithmetic statements.

The ROUNDED Phrase

When the result of an arithmetic operation has more digits to the right of the decimal point than the result-identifier provides for, low-order truncation occurs. The number of truncated digits is determined by the PICTURE clause of the result-identifier. When you specify the ROUNDED phrase and the first digit of the result's truncated portion is greater than 4, the value of the low-order digit in the result-identifier is increased by one. Conversely, when the first digit of the truncated portion is 4 or less, the digits are dropped. In either case, only the value of the first digit of the truncated portion is evaluated.

If the low-order integer positions in a result-identifier are represented by Ps in the PICTURE clause, the rightmost integer position for which storage is allocated is affected by rounding or truncation. For example, if the value 6853 is stored without rounding in an item with PICTURE 99PP, the value 6800 is stored in the result-identifier; if rounding is specified, the value stored is 6900. The examples in Table 1–9 show the results when using the ROUNDED phrase.

Table 1–9 Results with the ROUNDED Phrase

Picture	Amount	Result
9999	1234.49	1234
9999	1234.50	1235
999V99	123.995	124.00
9V9	9.375	9.4
99V9	50.11	50.1
999PP	12345	12300
999PP	67891	67900

The SIZE ERROR Phrase

A size error condition exists when, after decimal point alignment, the absolute value of a result is greater than the largest value that can be contained in the result-identifier. The size error condition applies both to the intermediate and to the final results of all arithmetic operations. If you specify the ROUNDED phrase, rounding occurs before checking for size error. Division by zero always causes a size error condition.

If a size error condition occurs during the performance of an arithmetic statement, the data-item affected by the statement retains its original value. If you specify SIZE ERROR *imperative-statement,* the imperative statement is executed after the current statement is completed. Without a SIZE ERROR phrase, the arithmetic statement is, in effect, bypassed.

The CORRESPONDING Phrase

The CORRESPONDING phrase processes all subordinate data-items having the same names within two group items. Two data-items correspond if they have the same data names and qualifiers up to, but not including, the group data-item mentioned in the statement. Corresponding data-items cannot be designated as USAGE IS INDEX, FILLER, OCCURS, or level 77 or 88. For details, see ADD CORRESPONDING, SUBTRACT CORRESPONDING, and MOVE CORRESPONDING in Chapter 8.

Conditional Expressions

The IF and PERFORM UNTIL statements specify conditional expressions. The program takes different actions depending on whether the condition is true or false. There are two categories of conditional expressions: simple conditions and complex conditions.

Simple Conditions

The simple conditions are the relation, condition-name, class, switch-status, and sign conditions. A simple condition has a value of true or false. Examples of simple conditions are:

Condition-name	`IF END-OF-FILE`
Class	`IF NAME IS NOT ALPHABETIC`
Switch-status	`IF SWITCH-IS-ON`
Sign	`IF COUNT IS NEGATIVE`
Relation	`PERFORM CHECK UNTIL A IS LESS THAN B`

Relation Conditions

A relation condition compares two items. Each item can be an identifier or a literal. The general format for a relation condition is:

```
                    ( GREATER THAN )
                    | LESS THAN    |
                    | EQUAL TO     |
                    |     >        |
id-lit IS [ NOT ]   <     <        >   id-lit
                    |     =        |
                    |     >=       |
                    (     <=       )
```

There are two types of comparisons: numeric and nonnumeric. If both items are numeric (that is, their PICTURE clauses contain only 9, S, V, or P), the comparison is numeric. If one or more items are nonnumeric, the comparison is nonnumeric.

Numeric Comparisons

Numeric comparisons are made according to algebraic value. The number of digits in an item does not affect the comparison. For example, 009 and 9 are equal. The following rules apply to numeric comparisons:

- The items can have different lengths and USAGE designations.
- Unsigned numeric items are considered positive.
- Zero is considered a unique value regardless of sign.

Nonnumeric Comparisons

If either item is a group item or is not numeric, a nonnumeric comparison takes place. A nonnumeric comparison is made according to the character collating sequence. Characters in corresponding positions are compared, starting from the high-order end. When the first unequal characters are encountered, the item that contains the character higher in the collating sequence is considered greater. Both items are equal if all comparisons are equal. The following rules apply to nonnumeric comparisons:

- The items being compared must be defined as USAGE IS DISPLAY.
- If the items have unequal lengths, comparison is made as if the shorter item were extended to the right with enough spaces to make both equal in length.
- When a figurative constant (except zero) is compared to a numeric identifier, the identifier must be defined as USAGE IS DISPLAY.

If one item is numeric:

- It must be an integer.
- It is treated as if it were moved to an elementary or group alphanumeric data-item of the same size as the nonnumeric data-item (depending on whether the nonnumeric data-item is an elementary or group item), and the contents of the alphanumeric item is used in the comparison.

The following example illustrates a relation comparison:

```
WORKING-STORAGE SECTION.
01  NONNUMERIC-GROUP.
    05  SUBGROUP-1.
        10  ITEM-1       PIC XX VALUE "aa".
        10  ITEM-2       PIC XX VALUE "aa".
        10  ITEM-3       PIC XX VALUE "aa".
    05  SUBGROUP-2       PIC XXX VALUE "aaa".

01  NUMERIC-ITEM  PIC 9(9) VALUE 111111111.

PROCEDURE DIVISION.

MAIN-PARAGRAPH.
    IF NONNUMERIC-GROUP IS >= NUMERIC-ITEM PERFORM SHOW-RESULTS.
    STOP RUN.

SHOW-RESULTS.
    DISPLAY "The nonnumeric group is >= numeric item.".
    DISPLAY "NONNUMERIC GROUP = ", NONNUMERIC-GROUP.
    DISPLAY "    NUMERIC ITEM = ", NUMERIC-ITEM.
```

When you run the program, the following appears on the screen:

```
The nonnumeric group is >= numeric item.

NONNUMERIC GROUP = aaaaaaaaa

    NUMERIC ITEM = 111111111
```

Condition-Name Condition

A condition-name is a special way of writing a relation condition. It is used to improve program readability. A condition-name must have a level number of 88 and must be followed by a VALUE clause, which specifies the literal or literals that apply to that condition.

The format of a level 88 entry is:

88 *condition-name* { VALUE IS | VALUES ARE } *lit-1* [{ THROUGH | THRU } *lit-2*]

[,*lit-3* [{ THROUGH | THRU } *lit-4*]] ...

Within a range, *lit-1* must be less than *lit-2* and *lit-3* less than *lit-4*. If the Working-Storage Section contains the following code:

```
05  HOURS-WORKED          PIC 99.
    88 REG-HOURS              VALUES ARE 1 THRU 40.
    88 OVERTIME-HOURS         VALUES ARE 41 THRU 99.
```

The Procedure Division could contain the following sentences:

```
IF REG-HOURS PERFORM REG-HOURS-ROUTINE.
IF OVERTIME-HOURS PERFORM OVERTIME-HOURS-ROUTINE.
```

The first sentence evaluates to true if the value of HOURS-WORKED is from 1 to 40. The REG-HOURS-ROUTINE is then performed. Similarly, the second sentence evaluates to true if the value of HOURS-WORKED is from 41 to 99. The OVERTIME-HOURS-ROUTINE is then performed.

Class Condition

The class condition test determines whether an item is alphabetic or numeric. An item is alphabetic if it consists entirely of uppercase letters A–Z, spaces, or any combination of these characters.

An item is numeric if its PICTURE clause does not contain an S and it contains digits 0–9, or if its PICTURE does contain an S and it contains digits 0–9 and a valid sign in the correct position. A numeric item's PICTURE clause can contain only the symbols S, 9, V, and P. Valid signs for data-items described with the SIGN IS SEPARATE clause are the plus and the minus signs. Valid signs for data-items not described with the SIGN IS SEPARATE clause are specified in the "SIGN Clause" section in Chapter 7.

The general format for the class condition is:

id IS [NOT] { NUMERIC | ALPHABETIC }

The following rules apply to class conditions:

- The item tested must be described as USAGE IS DISPLAY.
- The ALPHABETIC test cannot be used with a numeric item.
- The NUMERIC test cannot be used with an alphabetic item or a group item that contains a signed elementary item (PIC S).

Switch-Status Condition

A switch-status condition determines the on or off status of a switch defined in the Environment Division. The switch-name and the condition-name associated with the switch must be named in the SPECIAL-NAMES paragraph. The general format for the switch-status condition is:

IF *condition-name imper-stmt*

where *condition-name* is the literal defined in an ON or OFF clause in the SPECIAL-NAMES paragraph of the Environment Division. The condition-name need not be defined in Working-Storage. For example:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
...
SPECIAL-NAMES.
        SWITCH "A", ON STATUS IS SWO.
    ...
PROCEDURE DIVISION.
    ...
        IF SWO GO TO NEW-FILE-RTN.
```

The result of the test is true if the program is executed or chained to with the same local switch (in this case, A) declared in the program. To set a switch, append the switch to the program-name when you call the program. You can set up to 26 switches at runtime. See your user's guide for more information about setting switches.

Sign Condition

The sign condition determines whether the algebraic value of a numeric item is greater than, less than, or equal to zero. The general format for a sign condition is:

id IS [NOT] { POSITIVE | NEGATIVE | ZERO }

Complex Conditions

Complex conditions result from combining conditions with the logical operators NOT, AND, and OR, and using optional parentheses to indicate the order of evaluation. Table 1–10 lists the logical operators and their meanings.

Table 1-10 Logical Operators

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

Complex conditions occur in two forms: the negated simple condition and the combined condition. The general form for a negated simple condition is:

NOT *simple-condition*

The general format for a combined condition is:

condition [{ AND | OR } *condition*] ...

where *condition* can be a simple condition, negated simple condition, combined condition, or negated combined condition.

The following example illustrates a negated simple condition:

```
NOT TOT IS LESS THAN MAX
```

The following examples illustrate combined conditions:

```
TOT IS LESS THAN MAX OR PART IS LESS THAN WHOLE
TOT IS LESS THAN 0 OR PART IS GREATER THAN 100
```

The truth value of a complex condition is determined by first evaluating the simple conditions and then evaluating the conditions formed by the truth values and the logical operators. Each logical operator must be preceded and followed by a space.

Abbreviated Combined Relation Conditions

If a condition is combined, any condition except the first can be abbreviated by omitting either the subject of the relation condition or the subject and relational operator of the relation condition.

The format for an abbreviated combined relation condition is:

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

The following conditions are equivalent:

```
PAYROLL IS LESS THAN 0 OR PAYROLL IS LESS THAN TOTAL
PAYROLL IS LESS THAN 0 OR LESS THAN TOTAL
PAYROLL IS LESS THAN 0 OR TOTAL
```

In an abbreviated combined relation condition, if the word following NOT is GREATER, >, LESS, <, EQUAL, =, >=, or <=, then NOT participates as part of the relational operator. Otherwise, NOT is interpreted as a logical operator and, therefore, the implied insertion of subject or relational operator results in a negated relation condition. For example:

```
IF TYPE NOT = "O" AND NOT "S" AND NOT "W"
```

is equivalent to

```
IF (TYPE NOT = "O") AND (NOT TYPE = "S") AND (NOT TYPE = "W").
```

Also, ZERO is treated as a figurative constant instead of a sign condition. For example, the condition:

```
IF A < B AND ZERO
```

is interpreted as

```
IF A < B AND A < ZERO
```

Examples of abbreviated combined and negated combined relation conditions follow.

Condition	Expanded Equivalent
a > b AND NOT < c OR d	((a > b) AND (a NOT < c)) OR (a NOT < d)
a NOT EQUAL b OR c	(a NOT EQUAL b) OR (a NOT EQUAL c)
NOT a = b OR c	(NOT (a = b)) OR (a = c)
NOT (a GREATER b OR < c)	NOT ((a GREATER b) OR (a < c))

Evaluation Order of Conditions

When parentheses are not used or not nested, conditions are evaluated in the following order:

1. Simple conditions, in the following order: relation, class, condition-name, switch-status, sign
2. Negated simple conditions
3. Combined conditions with the logical operator AND
4. Combined conditions with the logical operator OR
5. Negated combined conditions

Use parentheses to change the order of evaluation. Conditions within parentheses are evaluated first. Within nested parentheses, evaluation proceeds from the innermost pair of parentheses to the outermost pair. When the sequence of evaluation is not completely specified by parentheses, the order of evaluation is from left to right in the sequence given above.

Parentheses are unnecessary when you use either AND or OR exclusively in one combined condition. However, you may need parentheses to determine a final truth value when the condition includes a combination of AND, OR, and NOT.

File I/O Operations

I/O operations transfer data to and from files and devices. The following Interactive COBOL verbs are used for I/O: ACCEPT, CLOSE, DELETE, DISPLAY, OPEN, READ, REWRITE, START, UNDELETE, UNLOCK, and WRITE.

Current Record Pointer

The current record pointer is a conceptual item indicating a logical position in an indexed or relative file in dynamic access mode. It is used to select the next record to be read. The current record pointer is changed by the OPEN, START, and READ statements. See these statements in Chapter 8 for more information.

File Status Codes

After an I/O operation is executed, the system generates a File Status code indicating the outcome of the operation. If you specify the FILE STATUS clause in a file control entry, the status value of the I/O operation is placed into the specified two-character data-item. Your program can then test this item to determine the condition that terminated the I/O operation.

File Status codes can be examined anywhere within the Procedure Division. However, special rules apply for examining File Status codes generated by an unsuccessful OPEN or CLOSE.

If an OPEN or CLOSE file statement is unsuccessful, your Interactive COBOL program terminates immediately unless your program includes a Declaratives section with a USE statement for which the file qualifies. The statement examining File Status codes is not executed if you do not code a USE statement.

Each user's guide lists the File Status codes for that operating system.

Exception Conditions

The following are examples of exception conditions that could arise during I/O processing:

- Specifying an invalid key for a random access file
- Reaching end of file during sequential processing
- Opening a file with an illegal filename
- Encountering device or data errors

Interactive COBOL provides two methods of handling these exceptions. You can specify the AT END and INVALID KEY options in the appropriate I/O processing statements for a file. You can also specify the Declaratives section at the beginning of the Procedure Division to process all exceptions for any files or sets of files.

The INVALID KEY Condition

The INVALID KEY condition occurs as the result of an unsuccessful execution of a DELETE, READ, REWRITE, START, UNDELETE, or WRITE statement. Three examples of invalid key errors are:

- A READ, REWRITE, or DELETE statement that attempts to access a record with a nonexistent primary file key
- A WRITE statement executed for an existing primary key
- A nonnumeric key value specified for a relative file

When the system recognizes the INVALID KEY condition, the following actions take place:

1. A nonzero 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 present, control passes to the associated imperative statement. Any USE procedure specified for the file is not executed.
3. If the INVALID KEY phrase is not present, but a USE procedure is specified for the file, that procedure is executed.

If the INVALID KEY condition occurs, the I/O statement that recognizes the condition fails, and the file is not affected.

The AT END Condition

The AT END condition occurs when a sequential READ, READ NEXT, or READ PREVIOUS statement attempts to process past the end of a file or before the beginning of a file. When the system recognizes the AT END condition, the following actions take place:

1. A nonzero value is placed in the FILE STATUS data-item, if specified for this file, to indicate an AT END condition.
2. If the AT END phrase is present, control passes to the specified imperative statement. Any USE procedure specified for the file is not executed.
3. If the AT END phrase is not present, but a USE procedure is specified for the file, that procedure is executed.

If the AT END condition occurs, the I/O statement that causes the condition fails. The contents of the associated record area and the status of the current record pointer are undefined.

See the READ statement in Chapter 8 for more information.

Transfer of Program Control

The run unit handles the transfer of control between programs based on whether you issue a CALL PROGRAM statement or a CALL statement. When you issue a CALL PROGRAM statement, the called program becomes the only program in the run unit. When you issue a CALL statement, however, control is transferred to a program currently in the run unit, or to a program that is brought into the run unit. The CALL statement does not clear the run unit. See Figure 1–1 for an example of how program control is passed between programs in a run unit.

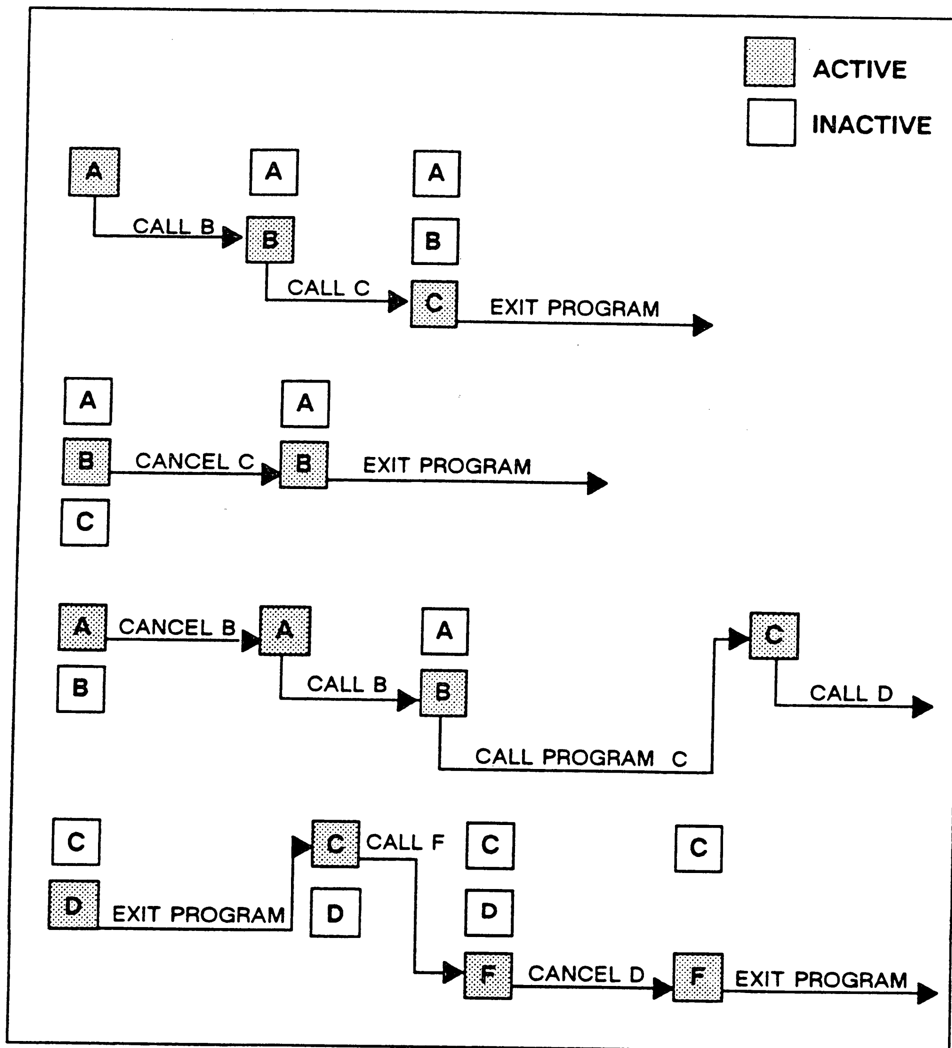


Figure 1–1 Transfer of Control Between Programs in a Run Unit

The CANCEL statement removes a program from the run unit.

See Chapter 8 for more information about verbs that transfer program control.

End of Chapter

Chapter 2
File Organization and Access

Interactive COBOL supports sequential, indexed, and relative file organizations. Files are automatically created by the system when a file that does not exist is opened for OUTPUT by a COBOL program, or when a relative or indexed file is opened for I-O. Table 2-1 lists file organizations and their associated record and key lengths.

Table 2-1 File Organizations, Record Lengths, and Key Lengths

File Organization	Maximum Record Size	Key Length
Sequential fixed or variable (assigned to DISK)	limited only by program size	n/a
line (assigned to PRINTER, KEYBOARD, or DISPLAY)	254 bytes (UNIX, AOS/VS, MS-DOS) 132 bytes (DG/RDOS, AOS)	n/a
Indexed	4096 bytes	100-byte max.
Relative	4096 bytes	2-byte binary

Use the FILECALC utility (UNIX®, AOS/VS, and MS-DOS® systems) or the FILESTATS utility (DG/RDOS and AOS systems) to determine the number of blocks required for an indexed or relative file. The utility uses number of records, key length, and record length as input, and provides index size, data size, and total file size as output. See your utilities manual for more information.

Sequential Files

The records in a sequential file are arranged in the order written. You can retrieve the records only in that order. You can read a sequential file only from the beginning of the file. If you add a record, it is appended to the file.

Sequential files have no record keys. A record in a sequential file cannot be deleted; a new file must be created to accomplish this.

There are three types of sequential files:

- Fixed sequential
- Variable sequential
- Line sequential

Sequential files are specified as ORGANIZATION IS SEQUENTIAL in the file-control entry in the Environment Division. You can omit this entry, since sequential organization is the default.

In fixed-length sequential files, each record is the same length. This length is determined at the time of file creation and cannot change. Record length is set with the RECORD CONTAINS clause in the Data Division. The RECORDING MODE IS FIXED clause specifies the file type as fixed sequential. You can omit this clause, since fixed sequential is the default.

In variable-length sequential files, each record contains a two-byte header that stores the record length. You must code a range of possible record sizes with the RECORD CONTAINS clause in the Data Division. The RECORDING MODE IS VARIABLE clause specifies the file type as variable sequential.

In fixed and variable sequential files, storage for the record is reserved in the program file. The size of the resulting program file cannot exceed the runtime limit for your system. See your user's guide for information on maximum program size.

Line sequential files written by Interactive COBOL programs contain variable length records terminated by an operating system specific delimiter. In a line sequential file, a data record is followed by the delimiter. Print-ready (or text) files are similar to line sequential files except that they can contain other carriage control characters as well as the terminator. Table 2–2 lists the file terminators for line sequential files.

Table 2–2 Terminators for Line Sequential Files

Terminators	Operating Systems
CR FF NULL	UNIX, AOS/VS, AOS, DG/RDOS, MS-DOS
NL	UNIX, AOS/VS, AOS, MS-DOS
CR/NL[1]	UNIX, MS-DOS

1. This sequence is treated as a single character.

Fixed and variable sequential files must be assigned to DISK, and line sequential files must be assigned to PRINTER or PRINTER-1 (for output files), KEYBOARD (for input files), or DISPLAY (to send data directly to the screen without using the Screen Section). Line sequential files do not use the RECORDING MODE IS clause.

You can specify a maximum record size by using the RECORD CONTAINS clause, but this is not required. There is a system limit on the maximum record size (see Table 2–1). Interactive COBOL writes out the number of bytes determined by the record you name in the WRITE statement. For example:

```
FD LINE-SEQ-FILE LABEL RECORD IS OMITTED
    RECORD CONTAINS 1 TO 80 CHARACTERS.
01 REC-OF-LENGTH-1      PIC X(1).
01 REC-OF-LENGTH-25     PIC X(25).
01 REC-OF-LENGTH-80     PIC X(80).
```

Using REC-OF-LENGTH-1 in a WRITE statement transfers one byte of data to the file plus any carriage control or terminator characters. Using REC-OF-LENGTH-25 transfers 25 bytes of data (padded with spaces, if necessary) plus carriage control or terminator characters. Using REC-OF-LENGTH-80 transfers 80 characters of data (padded with spaces, if necessary).

The usual method of entering data from the keyboard is using the ACCEPT verb and the Interactive COBOL Screen Section. An alternate method is assigning a line sequential file to KEYBOARD without specifying an external filename, and opening that file for input. Similarly, the usual method of putting data out to the terminal is using the DISPLAY verb and the Interactive COBOL Screen Section. An alternate method is assigning a line sequential file to DISPLAY without specifying an external filename, opening that file for output, and writing to it.

In a file with line sequential records, the READ statement inputs characters up to the first terminator or until the record is filled. Use the Ctrl-D sequence to produce an end of file condition for UNIX, AOS/VS, and AOS, and Ctrl-Z for MS-DOS and DG/RDOS.

When a READ statement is executed, the system overwrites the contents of the record area up to and including the terminator. The record area contains the record, the terminator, and any characters following the terminator that were present before execution of the READ. To avoid processing data from a previous record, clear the record area before reading a record. Do this by moving spaces or zeroes to the record area.

You can append records to a sequential file with the OPEN EXTEND statement. You can read, modify, and rewrite records consecutively with the OPEN I-O statement.

Indexed Files

An indexed file consists of records identified by key values, rather than by physical or logical position. The key allows direct access to a particular record using a single program statement. Indexed file organization is permitted only for files assigned to DISK.

An indexed file must have at least one key, and it can have up to one primary key and four alternate keys. The data-item named in the RECORD KEY clause of the SELECT clause for an indexed file is the primary record key for that file. The ALTERNATE RECORD KEY clause specifies the optional alternate keys. Both primary and alternate keys must be defined in a record description in the File Section of the Data Division. The definition consists of a level number, a data name (the name used in the SELECT clause), and a PICTURE clause. To insert, update, delete, and undelete records in a file, identify each record by the unique value of its primary key. The key value of HIGH-VALUES is reserved for system use only.

An indexed file is implemented as two external files at the operating system level. One file is the index portion and contains keys and pointers. The other file is the data portion and contains records. An Interactive COBOL program references the file by a single filename. The runtime system appends the extensions .NX and .XD, or .nx and .xd, to the filename to identify the index portion and the data portion, respectively.

The Index Portion

The index portion is logically organized in a tree structure, having a maximum of six levels. The root level consists of one node, which contains key entries in ascending collating sequence. Each key entry in the root level has the highest ASCII value in a node at the second level. Pointers connect each key entry in the root level to the appropriate node at the next level. Succeeding levels follow the same pattern until a terminal level is reached. Nodes in the terminal level contain key entries for all existing records. Associated with each key entry is a pointer to the data record in the data portion. Figure 2–1 shows the tree structure of an indexed file.

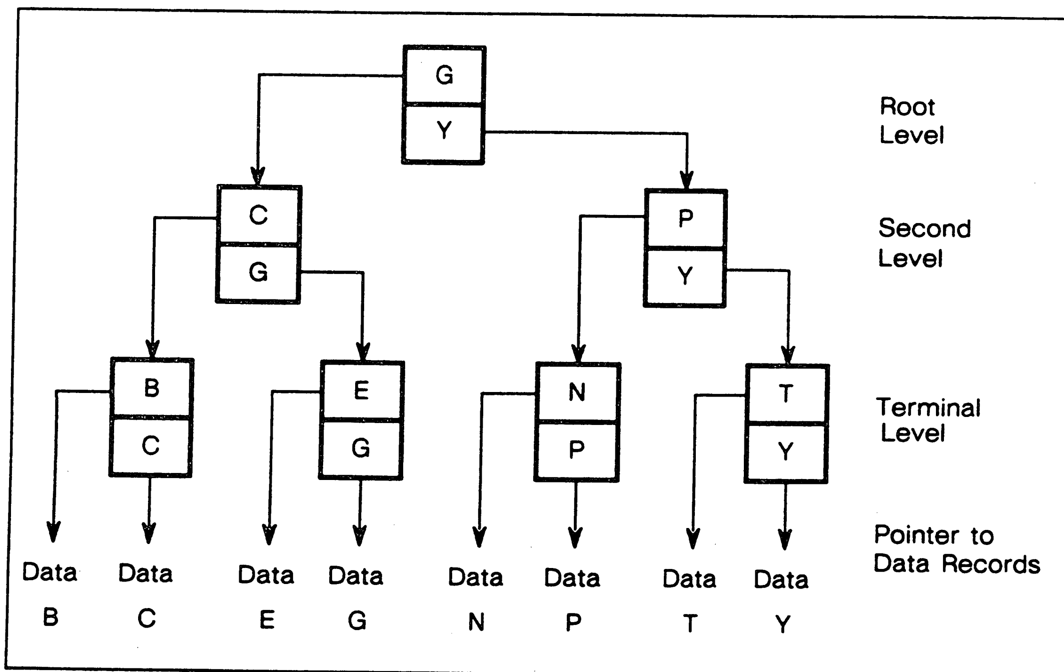


Figure 2–1 Indexed File Structure

The Data Portion

The data portion of an indexed file contains the data records, including the keys, in the order in which records are written; they are not necessarily in ascending collating sequence. This implementation makes file expansion simpler since file space on the disk is allocated as needed. Consequently, records need no overflow areas.

Sequential Access

The data portion of an indexed file is structured so that sequential access is efficient. Data records are linked within the data portion; thus, sequential access is independent of the index portion. In sequential access mode, records are written and retrieved in ascending order of record key values.

Random Access

In random access, the search algorithm compares a given key to the key entries in a node at one level of the structure and follows a pointer to a node at the next level. The index structure and the algorithm make the search efficient. For example, to

Licensed Material – Property of Data General Corporation

locate a record with record key E (refer to Table 2–3), E is compared to the first entry at the root level. Since E is less than G in the collating sequence, the search proceeds to the second level by following the pointer from G. The pointer leads to the block containing C and G. Since E is greater than C and less than G, the search continues to the third level by following the pointer from G. In this example, the third level is the terminal level, and the pointer to the data record is reached.

Dynamic Access

Dynamic access enables you to use one OPEN statement to access records sequentially or randomly in relative and indexed files. The ACCESS MODE clause, described in Chapter 6, is used to access files dynamically.

File Efficiency

As shown by the example of random access, the search through the index portion for a given key requires a "read and compare" at each level of the tree structure. Thus, one objective in designing a file is to minimize the number of levels in the index structure. Keys can be up to 100 bytes long, but shorter keys are generally more efficient. Longer keys mean fewer key entries per block, hence a greater number of levels in the tree structure. The number of levels also increases as the number of records increases.

Use the FILECALC or FILESTATS utility at the file design stage to predict the size of the index portion of a file with a given key length and a given number of records. The output of these utilities includes the number of key entries per index block and the number of levels in the structure, as well as the calculated sizes of the index portion and the data portion.

Record Deletion

The DELETE verb deletes records logically. The record in the data portion is flagged as deleted and, for the user, no longer exists. However, the data record is not physically removed from the file. Interactive COBOL provides the UNDELETE verb, which restores a logically deleted record to active use. DELETE and UNDELETE require the use of the primary record key.

The ANALYZE utility reports the number of records logically deleted from a file. If this number becomes large in relation to the total number of records, disk space is wasted, and the index portion contains more levels than necessary. The COLLAPSE utility (DG/RDOS, AOS), the ICOMPACT utility (UNIX, AOS/VS, MS-DOS), and the REORG utility physically delete logically deleted records. See your utilities manual for more information.

Primary and Alternate Keys

The primary key must uniquely identify a record. An alternate key can have duplicates; several records may have the same value for an alternate key. In addition to a primary key, you can specify up to four alternate keys. All keys must be alphanumeric.

An indexed file can be designed so that a record can be accessed by more than one path. For example, a file containing sales force records could have the salesperson's

name as the primary key and the salesperson's employee number and sales region as alternate keys. The sales region key can be used to retrieve the names of all the sales personnel in a given region.

When duplicate alternate keys are created, access paths to the records are built in the order of entry to the file. Therefore, when an alternate key with duplicates is used to retrieve a record, the first duplicate written is the first to be read.

An alternate key can also serve as another unique record identifier. However, if an alternate key will be a unique reference, the applications program must check the uniqueness of the key, because COBOL will not. The INVALID KEY phrase controls checking for duplicates of the primary key only.

Alternate keys are named in ALTERNATE RECORD KEY clauses in the SELECT clause and defined along with the primary key in a record description in the Data Division. The definition consists of a level number, a data name (the same as used in the SELECT clause), and a PICTURE clause. Each key can be up to 100 bytes long, with a maximum total of 500 bytes for a primary key and four alternates.

At the file system level, the index portion of an indexed file with alternate keys contains index structures for the primary key and for each alternate key. All index structures are the same as the structure described in Figure 2-1. Each index structure points to a common data portion. At the index's terminal level for an alternate key, each key entry points to the first data record that contains that alternate key value. Once the first record is located, additional records with that value for the alternate key can be accessed sequentially.

File Updating

A READ or START statement can refer to an alternate key, but a WRITE or REWRITE statement must refer to a primary key. When you execute a WRITE statement, the primary and alternate keys are added to their respective index structures.

A REWRITE statement can change an alternate key, but not a primary key. The record update takes longer for a file with alternate keys. When you change an alternate key, the system flags the old index entry for deletion and writes a new index entry. The data portion is merely updated in place. Therefore, if you run ANALYZE on a file after rewriting a large number of alternate keys, ANALYZE may report a number for deleted keys that is greater than the total number of records (including deletions) currently in the file. See your utilities manual for a description of ANALYZE.

Licensed Material - Property of Data General Corporation

Relative Files

A relative file consists of records and a two-byte binary key that uniquely identifies the logical ordinal position of each record in the file. The key is not part of the data record. The organization of relative files is similar to that for indexed files. A relative file is also implemented as two files, an index portion and a data portion, named with the extensions .NX and .XD, or .nx and .xd. The index portion has the tree structure described for indexed files.

The advantage of using a relative file is that it has a two-byte key, described as PIC 9(4) COMP to PIC 9(18) COMP. The key can identify up to 65,535 records. Five bytes per key would be required for this many records if the key were in decimal format in an indexed file. The reduced key length means that relative files have fewer index levels than indexed files; hence, access time is improved. Relative files also provide all the advantages of indexed access when keys can be expressed as numbers from 0 to 65,534. The data portion is the same as that for indexed files, except that the key is not part of the record.

If you use the switch that produces ANSI-compatible code, PIC 9(4) COMP allows you to access relative keys with values from 0 to 9999 only. To access keys with values from 10,000 to 65,534, define the relative key as PIC 9(5) COMP. If you do not use this switch, defining the relative key as PIC 9(4) COMP allows you to access all records (keys with values from 0 to 65,534).

The FILECALC or FILESTATS utility helps you design relative files. The ANALYZE utility supplies statistics for existing relative files.

End of Chapter

Chapter 3
Interactive Screen Management

With Interactive COBOL's screen management facility, an Interactive COBOL program can:

- Specify exact display locations for input, output, and update fields.
- Display literal text in predefined display positions.
- Accept data typed in designated positions on the display.
- Update the values of fields on the display.
- Control such console features as blinking or sounding a tone.
- Control such terminal features as the intensity of the display and the foreground and background colors when you are using code revision 9 or greater.

A single program can contain many different screens. The Screen Section of the Data Division defines screen entries, and the DISPLAY and ACCEPT verbs refer to these entries.

Screen Data Description

Each field that appears on the display must be described in the Screen Section. Each screen entry consists of a level number, an optional screen-name, and optional clauses that specify the positions of display fields and other display terminal functions, such as blinking or sounding a tone.

Screen entries can be literals, data-items, or group items. Literal format is used to display constant information, such as a heading or a menu. Data-item format is used to pass data between File, Working-Storage, and Linkage sections and screen storage. Literal and data-item entries can be combined to form a group entry, which allows you to display and accept an entire screen by executing one DISPLAY and ACCEPT sequence.

The following rules apply to the Screen Section:

- The Screen Section must follow the Working-Storage and Linkage sections.
- The level number must be the first element in each data description. Level numbers range from 01 to 49. Entries at the 01 level must begin in area A; other levels can be indented as desired.
- A screen-name is required with 01 level numbers but not with other level numbers. If FILLER is used, it must be the first word following the level number in each screen description entry. Screen–name assigns a name to the screen item described in the screen description and must conform to the rules for user–defined names. While you can use the keyword FILLER to name a screen item, you cannot refer to the FILLER item explicitly.

- If you omit the FILLER or screen-name clause, the system treats the screen item being described as though FILLER had been specified.
- You cannot use the REDEFINES and OCCURS clauses in the Screen Section.

Literal Format

A literal entry is typically a line of text, although it can be a single character or simply positioning information. The syntax for a literal entry is:

```
level-number [screen-name]
    [; BLANK SCREEN ]
    [; BLANK LINE ]
    [; BELL ]
    [; BLINK ]
    [; ERASE EOL ]
    [; ERASE EOS ]
    [; ERASE LINE ]
    [; REVERSE-VIDEO ]
    [; UNDERLINE ]

    [ ; { LOWLIGHT  } ]
    [   { HIGHLIGHT } ]
    [   { DIM       } ]
    [   { BRIGHT    } ]

    [ ; LINE NUMBER IS { integer-1         } ]
    [                  { PLUS integer-2    } ]
    [                  { identifier-1      } ]

    [ ; { COLUMN } NUMBER IS { integer-1      } ]
    [   { COL    }           { PLUS integer-2 } ]
    [                        { identifier-1   } ]

    [ ; FOREGROUND-COLOR IS { integer-3 } ]
    [                       { BLACK     } ]
    [                       { BLUE      } ]
    [                       { GREEN     } ]
    [                       { CYAN      } ]
    [                       { RED       } ]
    [                       { MAGENTA   } ]
    [                       { BROWN     } ]
    [                       { WHITE     } ]

    [ ; BACKGROUND-COLOR IS { integer-4 } ]
    [                       { BLACK     } ]
    [                       { BLUE      } ]
    [                       { GREEN     } ]
    [                       { CYAN      } ]
    [                       { RED       } ]
    [                       { MAGENTA   } ]
    [                       { BROWN     } ]
    [                       { WHITE     } ]

    [; VALUE IS literal].
```

A literal entry consists of a level number, an optional screen-name, and at least one other clause. If you display a literal, it must be a quoted string. The string can contain any text, such as a report heading, a prompt requesting information, or an error message. The following example shows a literal entry. Note that the VALUE IS clause is not required when specifying a literal.

```
01  NAME-PROMPT LINE 5 COL 5 "ENTER CUSTOMER NAME:".

PROCEDURE DIVISION.
    ...
    DISPLAY NAME-PROMPT.
```

When you execute the DISPLAY statement, the prompt appears on line 5, column 5. However, the operator cannot respond to the prompt, because literal format lets you display constant information only. You cannot execute an ACCEPT statement on an entry in literal format. To let the operator enter information, you must use data-item or group format.

Data-Item Format

The syntax for a data-item entry is:

```
level-number [screen-name]
    [; BLANK SCREEN ]
    [; BLANK LINE ]
    [; BELL ]
    [; BLINK ]
    [; ERASE EOL ]
    [; ERASE EOS ]
    [; ERASE LINE ]
    [; REVERSE-VIDEO ]
    [; UNDERLINE ]
    [; { LOWLIGHT  } ]
    [  { HIGHLIGHT } ]
    [  { DIM       } ]
    [  { BRIGHT    } ]
    [; LINE NUMBER IS { integer-1          } ]
    [                 { PLUS integer-2     } ]
    [                 { identifier-1       } ]
    [; { COLUMN } NUMBER IS { integer-1      } ]
    [  { COL    }           { PLUS integer-2 } ]
    [                       { identifier-1   } ]
    [; FOREGROUND-COLOR IS { integer-3 } ]
    [                      { BLACK     } ]
    [                      { BLUE      } ]
    [                      { GREEN     } ]
    [                      { CYAN      } ]
    [                      { RED       } ]
    [                      { MAGENTA   } ]
    [                      { BROWN     } ]
    [                      { WHITE     } ]
    [; BACKGROUND-COLOR IS { integer-4 } ]
    [                      { BLACK     } ]
    [                      { BLUE      } ]
    [                      { GREEN     } ]
    [                      { CYAN      } ]
    [                      { RED       } ]
    [                      { MAGENTA   } ]
    [                      { BROWN     } ]
    [                      { WHITE     } ]
    [; { PICTURE } IS picture-string { ; TO id [ FROM id-lit ] } ]
    [  { PIC     }                   { ; USING id              } ]
    [                                { ; FROM id-lit [ TO id ] } ]
    [; BLANK WHEN ZERO ]
    [; { JUSTIFIED } RIGHT ]
    [  { JUST      }       ]
    [; AUTO ]
    [; SECURE ]
    [; REQUIRED ]
    [; FULL ].
```

Screen section clauses can occur in any order.

An entry in data-item format must contain a PICTURE clause, which describes how the system edits the value of the data-item, and a FROM, TO, or USING clause, which specifies a corresponding File, Working-Storage, or Linkage Section data-item.

The FROM clause specifies an output data-item. During execution of a DISPLAY statement, the value of the data-item is displayed.

The TO clause specifies an input data-item. During execution of a DISPLAY statement, underscores are displayed according to the data-item's PICTURE clause. The underscores indicate the number of character positions in the display field. During execution of an ACCEPT statement, the operator can alter the value of the data-item.

The USING clause specifies an update data-item. During execution of a DISPLAY statement, the current value of the data-item is displayed. During execution of an ACCEPT statement, the operator can alter that value.

The other clauses define the position of the data-item in the display, and any special terminal functions; they are discussed later in this chapter.

The following example illustrates the pertinent statements for a simple screen I/O operation:

```
WORKING-STORAGE SECTION.
    ...
    05  CUST-NAME    PIC X(25).
    ...
SCREEN SECTION.

01  NAME-PROMPT LINE 5 COL 5 "ENTER CUSTOMER NAME:".
01  SCR-NAME LINE 5 COL 27 PIC X(25) TO CUST-NAME.
    ...

PROCEDURE DIVISION.
    ...
    DISPLAY NAME-PROMPT.
    DISPLAY SCR-NAME.
    ACCEPT SCR-NAME.
```

In the above example, NAME-PROMPT is a literal entry and SCR-NAME is a data-item entry. Execution of the first DISPLAY statement displays the value of NAME-PROMPT. Execution of the second DISPLAY statement displays 25 underscores. The ACCEPT statement allows the operator to enter data for the SCR-NAME entry. When the ACCEPT is complete, the values in screen storage are transferred to the data-item CUST-NAME in the Working-Storage Section.

Group Format

A group entry consists of one or more data-item or literal entries, or a combination of both. The syntax for a group entry is:

```
level-number [ { screen-name } ]
               { FILLER      }

    [; BLANK SCREEN ]
    [; FULL ]
    [; AUTO ]
    [; BLINK ]
    [; SECURE ]
    [; REQUIRED ]
    [; REVERSE-VIDEO ]
    [; UNDERLINE ]
    [   { LOWLIGHT  } ]
    [ ; { HIGHLIGHT } ]
    [   { DIM       } ]
    [   { BRIGHT    } ]

    [                                 { integer-1 } ]
    [                                 { BLACK     } ]
    [                                 { BLUE      } ]
    [                                 { GREEN     } ]
    [ ; FOREGROUND-COLOR IS           { CYAN      } ]
    [                                 { RED       } ]
    [                                 { MAGENTA   } ]
    [                                 { BROWN     } ]
    [                                 { WHITE     } ]

    [                                 { integer-2 } ]
    [                                 { BLACK     } ]
    [                                 { BLUE      } ]
    [                                 { GREEN     } ]
    [ ; BACKGROUND-COLOR IS           { CYAN      } ]
    [                                 { RED       } ]
    [                                 { MAGENTA   } ]
    [                                 { BROWN     } ]
    [                                 { WHITE     } ]

    [; [USAGE IS ] DISPLAY]
    [; [SIGN IS ] { LEADING  } SEPARATE CHARACTER
                  { TRAILING }
    { data-item     } .
    { literal entry }
```

With a group entry, you can display or accept multiple screen entries by executing a single DISPLAY or ACCEPT statement. You can execute an ACCEPT statement on a group entry that contains data-item or literal entries, as long as one or more entries

contain the TO or USING clause. You cannot execute an ACCEPT statement on an entry that includes only FROM clauses and literal entries.

A group entry is completed after you press a terminator at the last input or update field and the data is validated. Until that time, you can use the arrow keys to move to any input or update field in the display and modify it.

If you include the AUTO, FULL, BLINK SECURE, REQUIRED, LOWLIGHT, HIGHLIGHT, DIM, BRIGHT, UNDERLINE, REVERSE-VIDEO, FOREGROUND-COLOR, BACKGROUND-COLOR, or SIGN clause at a group level, it applies to each input and update field in the group. If you specify the same clause more than once in a group item, the clause that is lowest in the hierarchy is the one that takes affect.

A group entry is constructed according to the usual COBOL rules. It can contain a hierarchy of other group and elementary items, with a maximum of 49 levels. Level numbers can range from 01 to 49. The following example illustrates group format:

```
DATA DIVISION.
    ...
WORKING-STORAGE SECTION.
77  CUST-NAME       PIC X(25).
77  CUST-NUMBER     PIC 9(8).
77  CHARGE-AMOUNT   PIC 999V99.
    ...
SCREEN SECTION.
01  CUST-DATA.
    02  LINE 5  COL 5   "ENTER CUSTOMER NAME:".
    02  LINE 5  COL 33  PIC X(25) USING CUST-NAME.
    02  LINE 7  COL 5   "ENTER ACCOUNT NUMBER:".
    02  LINE 7  COL 33  PIC 9(8) USING CUST-NUMBER.
    02  LINE 9  COL 5   "ENTER AMOUNT OF CHARGE:".
    02  LINE 9  COL 33  PIC $999.99 USING CHARGE-AMOUNT.
    ...
PROCEDURE DIVISION.
    ...
    DISPLAY CUST-DATA.
    ACCEPT CUST-DATA.
```

A group entry must have a name; however, you can omit names for items within the group, as shown above. Omitting a name is equivalent to coding a FILLER entry (FILLER is illegal in the Screen Section after an 01 item). You cannot reference the individual entry except as part of a higher level group.

Screen Section Clauses

Interactive COBOL provides an extended set of clauses for the Screen Section. More than one clause can be associated with each screen entry. Clauses can be in any order within an entry; however, certain clauses cannot be used together in a screen definition. Table 3-1 lists the clauses you can use with each format. This section describes each clause.

NOTE: The following attributes work only if you are using code revision 9 or greater: BRIGHT/HIGHLIGHT, DIM/LOWLIGHT, FOREGROUND-COLOR, BACKGROUND-COLOR, REVERSE-VIDEO, UNDERLINE, EOL, EOF, EOS, and JUSTIFIED.

Table 3-1 Screen Clauses

Group Format	Literal Format	Data-Item Format
AUTO[1]	BELL	AUTO[1]
FULL[1]	BLANK LINE	BELL
REQUIRED[1]	BLANK SCREEN	BLANK LINE
SECURE[1]	BLINK	BLANK SCREEN
BLINK	ERASE EOL	BLANK WHEN ZERO
BLANK SCREEN	ERASE EOS	BLINK
REVERSE-VIDEO	ERASE LINE	COLUMN
UNDERLINE	REVERSE-VIDEO	FROM[2]
LOWLIGHT	UNDERLINE	FULL[1]
HIGHLIGHT	HIGHLIGHT	JUSTIFIED
DIM	LOWLIGHT	LINE
BRIGHT	DIM	PICTURE IS
FOREGROUND-COLOR	BRIGHT	REQUIRED[1]
BACKGROUND-COLOR	LINE	SECURE[1]
SIGN	COLUMN	TO[2]
USAGE	FOREGROUND-COLOR	USING
DISPLAY	BACKGROUND-COLOR	ERASE EOL
LEADING	VALUE IS	ERASE EOS
TRAILING		ERASE LINE
FILLER		REVERSE-VIDEO
		UNDERLINE
		LOWLIGHT
		HIGHLIGHT
		DIM
		BRIGHT
		FOREGROUND-COLOR
		BACKGROUND-COLOR

1. AUTO, FULL, REQUIRED, and SECURE cannot be used for an entry designated as output only (FROM clause).
2. An entry can contain a FROM clause and a TO clause; however, if it includes either clause it cannot also specify the USING clause.

VALUE Clause

The VALUE clause specifies the literal that will be displayed. You can use it only in a literal entry. The VALUE literal must be nonnumeric and must be enclosed in quotation marks. It cannot be a figurative constant. The phrase VALUE IS is optional.

DISPLAY Clause

The DISPLAY clause specifies that the data is stored in ASCII format, which is the default. You can only use the DISPLAY clause in a group entry.

PICTURE Clause

The PICTURE clause functions in much the same way as in the other sections of the Data Division. It describes the category, size, and appearance of the display field.

The picture-string can include any standard editing characters, such as those for comma insertion, zero suppression, or floating currency sign. You can use the picture-string characters P, V, CR, and DB only with output fields (FROM clause).

The PICTURE clause controls the appearance of displayed data. During execution of a DISPLAY statement, data is displayed according to the PICTURE associated with the data-item. All editing characters are present and any special clauses are in effect.

The PICTURE clause also checks the category of data that is accepted in a field. When you terminate an input or update field, the runtime system checks whether the entered data conforms to the PICTURE of the screen data-item.

For numeric fields, the data is evaluated on a character by character basis. If you enter an alphabetic character in a numeric field, the system displays an error message, positions the cursor to the first incorrect character, and waits for you to re-enter the data.

For alphanumeric fields, including PICTURE clauses containing numeric and alphabetic characters (PIC 99A), the data is validated on a field basis. The data you enter for a PIC 99A field can contain alphanumeric characters without regard to the actual field description.

The PICTURE of the File, Working-Storage, or Linkage data-item named in the FROM, TO, or USING clause does not have to correspond exactly to the PICTURE clause in the screen data-item. During execution of a DISPLAY/ACCEPT sequence, data is transferred between the File, Working-Storage, or Linkage data-item and the screen data-item according to the usual rules for a MOVE statement, with one exception: you can move data from a numeric edited data-item in the Screen Section to a numeric data-item in another section. In this case, the system ignores all characters in the numeric edited data-item except digits, sign, and decimal point. The decimal point is used for alignment purposes only.

FROM, TO, and USING Clauses

Every data-item entry must include a FROM, TO, or USING clause. You must define the identifier following the FROM, TO, or USING clause in the File, Working-Storage, or Linkage section of the Data Division. The identifier can be qualified, subscripted, or indexed.

FROM Clause

A FROM clause identifies an output field. When you execute a DISPLAY statement, the values in the associated File, Working-Storage, or Linkage data-items are moved into the screen data-items and are displayed. Fields with a FROM clause appear at half intensity unless another attribute is specified.

TO Clause

A TO clause identifies an input field. The TO identifier refers to a data-item in the File, Working-Storage, or Linkage section. When you execute a DISPLAY statement for a screen data-item with the TO clause, the system displays underscores. The number of underscores displayed corresponds to the number of characters in the picture-string. You can also enter underscores as input characters; however, all trailing underscores will be truncated. After you complete the ACCEPT statement, the value of the data-item in screen storage moves to the associated File, Working-Storage, or Linkage data-item. Fields with a TO clause appear at full intensity unless another attribute is specified.

If you terminate an input field without entering data and no previous value is retained, or you do not access an input field, the system transfers a default value from the screen data-item to the associated File, Working-Storage, or Linkage data-item. This default value is zero for numeric data-items and spaces for other data-items.

USING Clause

A USING clause identifies an update field, a field that is both input and output. When you execute a DISPLAY statement, the current value of the screen data-item is displayed.

When you execute an ACCEPT statement, you can enter new data in the update field. The values of the display field are left-justified, and editing characters are removed. The remainder of the field is filled with underscores.

You cannot combine the USING clause with the FROM or TO clause in one entry. Fields with a USING clause appear at full intensity unless another attribute is specified.

A screen entry can specify that data will be displayed from one id or id-lit and accepted into another. This operation requires both a FROM and TO clause, rather than a USING clause. You can transfer data between the File, Working-Storage, or Linkage data-item and the screen data-item according to the usual rules for MOVE statements, with one exception: you can move data from a numeric edited screen data-item to a numeric data-item.

LINE and COLUMN Clauses

LINE and COLUMN clauses specify the position of the field in the display. When a screen item is accepted or displayed, the field position begins at the location

designated in the LINE and COLUMN clauses. The system overwrites any existing characters in that screen location.

The maximum display size for all systems is 255 columns and 255 lines. Most terminals have a display size of 80 columns and 24 lines. Since system error messages are displayed on line 24, keep this line free when designing displays. See your user's guide for information about setting the display size.

You can specify a display position absolutely or relatively. Relative positions are always set at compile time. Absolute positions are set at compile time, if LINE or COLUMN is specified by an integer, or at runtime, if LINE or COLUMN is specified by an identifier. The following sections discuss absolute and relative positioning.

Absolute Positioning Using an Integer

An absolute position indicates a specific line and column number. You can specify the line or column number with an unsigned integer or an identifier (discussed below). The first positioning clause in a group must always be absolute. For example:

```
01  HEADER-SCREEN.
    02  LINE 3 COL 5 VALUE "MONTHLY UPDATE REPORT".
```

Relative Positioning

The PLUS phrase specifies relative positioning. The positioning is relative to the last position of the cursor. This eliminates the need to count lines or columns. An example of relative positioning is:

```
01  HEADER-SCREEN.
    05  LINE 3 COL 5 VALUE "MONTHLY UPDATE REPORT".
    05  LINE PLUS 2 COL 3 VALUE "CUSTOMER NAME".
    05  LINE PLUS 1 COL 3 VALUE "-------- ----".
```

Relative display positions are translated to absolute positions by the compiler. All relative positions are absolute at compile time. The order in which you define relatively positioned data-items, rather than the order in which you display or accept the fields, determines their actual display positions. For example:

```
SCREEN SECTION.

01 SCREEN-PROMPTS.
   05  SCR-EMP-PROMPT LINE 5 COL 5 PIC X(20) FROM EMP-PROMPT.
   05  SCR-EMP-NUM LINE 5 COL 28 PIC 9(9) TO EMP-NUM.
   05  SCR-PAY-PROMPT LINE PLUS 2 COL 5 PIC X(20) FROM PAY-PROMPT.
   05  SCR-PAY-RATE COL 28 PIC 99.99 TO PAY-RATE.
   05  SCR-HRS-PROMPT LINE PLUS 2 COL 5 PIC X(20) FROM HRS-PROMPT.
   05  SCR-HRS COL 28 PIC 99.99 USING HRS.

   ...
   DISPLAY SCR-PAY-PROMPT.
   DISPLAY SCR-PAY-RATE.
   ACCEPT SCR-PAY-RATE.
   IF PAY-RATE < 50.00 PERFORM PAY-RATE-ERROR-PARA.
```

```
...
DISPLAY SCR-HRS-PROMPT.
DISPLAY SCR-HRS.
ACCEPT SCR-HRS.
IF HRS < 40.00 PERFORM OVERTIME-PARA.
```

The DISPLAY and ACCEPT sequence for pay rate and hours worked can appear in any order. The sequence does not affect the final appearance of the display, but does affect the order in which the operator responds to the prompts.

Each clause has four options:

- No LINE or COLUMN clause
- Only the LINE or COLUMN clause with no number
- The LINE or COLUMN clause with a number: LINE *m* or COLUMN *n*
- The LINE or COLUMN clause with the PLUS phrase: LINE PLUS *m* or COLUMN PLUS *n*.

Using one form of one clause can affect the other clause. Table 3-2 shows the resulting field position for each combination.

Table 3-2 LINE and COLUMN Positioning

LINE Clause	COLUMN Clause	Field Position
No clause	No clause	Current line, current col
	COL	Current line, column plus 1
	COL *n*	Current line, column *n*
	COL PLUS *n*	Current line, column plus *n*
LINE	No clause	Line plus 1, column 1
	COL	Line plus 1, column plus 1
	COL *n*	Line plus 1, column *n*
	COL PLUS *n*	Line plus 1, column plus *n*
LINE *m*	No clause	Line *m*, column 1
	COL	Line *m*, column plus 1
	COL *n*	Line *m*, column *n*
	COL PLUS *n*	Line *m*, column plus *n*
LINE PLUS *m*	No clause	Line plus *m*, column 1
	COL	Line plus *m*, column plus 1
	COL *n*	Line plus *m*, column *n*
	COL PLUS *n*	Line plus *m*, column plus *n*

Absolute Positioning Using an Identifier

Lines and columns can also be specified by an identifier, which allows positioning to be defined at execution time. The identifier must be an elementary unsigned item and can be qualified, subscripted or indexed. The following example shows a screen entry with dynamic LINE and COLUMN setting:

```
05 LINE LINE-NUM COL COL-NUM "ENTER CUSTOMER NAME: ".
```

If you specify a line number dynamically, you cannot specify any following lines in the screen entry relatively. That is, the phrase LINE PLUS *n* cannot occur after you use an identifier to specify a line. The LINE clause with no number is equivalent to LINE PLUS 1.

If you specify a column number dynamically, you cannot specify any following columns in the screen entry relatively. That is, the phrase COLUMN PLUS *n* cannot occur after you use an identifier to specify a column. The COLUMN clause with no number is equivalent to COLUMN PLUS 1.

Positioning Sequence

Line and column positioning is performed in the following order:

- Absolute position specified with integers (LINE *n*, COL *n*). These line numbers are set during compile time.
- Relative position (LINE PLUS *n*, COL PLUS *n*). These line numbers are set during compile time.
- Absolute position specified with identifiers (LINE *id*, COLUMN *id*). These line numbers are set during runtime.
- Variable origin for an entire display (DISPLAY *screen-name* AT, ACCEPT *screen-name* AT). The runtime system offsets all absolute and relative positions according to this origin.

BLANK SCREEN and ERASE EOS Clauses

The BLANK SCREEN clause erases the entire display and positions the cursor to line 1, column 1. The ERASE EOS clause clears the screen starting at the cursor position specified by the screen data item. It does not affect the screen display prior to that data item. (ERASE EOS works under code revision 9 or greater.)

BLANK LINE, ERASE EOL, and ERASE LINE Clauses

The BLANK LINE and ERASE EOL clauses erase the current line from the cursor position to the end of the line without changing the cursor position. ERASE LINE erases the current line beginning at column one continuing through the end of the line. Since the system executes the BLANK LINE or ERASE EOL clause after the positioning commands LINE and COLUMN, you can use these clauses as well as ERASE LINE to clear a previously displayed item (such as an error message) before displaying a new item at the same position. (ERASE EOL works under code revision 9 or greater.)

BACKGROUND-COLOR and FOREGROUND-COLOR Clauses

The BACKGROUND-COLOR clause determines the background color for a screen item while the FOREGROUND-COLOR clause determines the foreground color for the item. These clauses are effective only with color screens. You specify the color by entering an integer from 0 to 7 or the appropriate keyword. These are:

0 BLACK
1 BLUE
2 GREEN
3 CYAN
4 RED
5 MAGENTA
6 BROWN
7 WHITE

When you use these clauses on a group level, they apply to all the subordinate screen items. If you do not specify these colors, the terminal uses its default background and foreground colors. You can only use these attributes when you are working with code revision 9 or greater.

NOTE: The BRIGHT/HIGHLIGHT and DIM/LOWLIGHT clauses have no affect on the background color. The system always sets the background color to the dim version of a color.

BELL Clause

The BELL clause sounds the terminal tone. It is used to attract your attention. Use the BELL clause with literal or data-item formats. If the BELL clause is part of a group entry, the tone sounds when the group is displayed.

Input Control Clauses

The input control clauses affect input fields during data entry. They restrict the way in which you enter data, the way data echoes on the screen, and the way fields terminate. These clauses are AUTO, FULL, REQUIRED, and SECURE.

Input control clauses in a group entry are associated only with input or update items in the group (items having the TO or USING clause).

The AUTO clause terminates the input or update field when it is full, and advances automatically to the next field. If only one entry will be input, or if the entry is the last one in a group, execution of the ACCEPT statement is completed.

The FULL clause requires that, if you enter data, you must enter a character or space in each position before terminating the input or update field. PIC X, PIC A, and PIC 9 data-items accept spaces. USING fields are initially always full. For a TO field, pressing only a terminator bypasses the field.

The REQUIRED clause specifies that the operator cannot bypass the input or update field. The field must contain at least one character.

The SECURE clause echoes asterisks rather than actual data on the screen. This protects the privacy of such data as authorization codes. When you specify this clause, the JUSTIFIED and BLANK WHEN ZERO clauses have no effect.

Display Clauses

Display clauses define the way in which a literal or data-item field is displayed, and the way in which entered data is redisplayed after a data-item field is accepted. These clauses are BLANK WHEN ZERO, BLINK, JUSTIFIED, BRIGHT, HIGHLIGHT, DIM, LOWLIGHT, REVERSE-VIDEO, and UNDERLINE.

The BLANK WHEN ZERO clause fills the field with spaces when the data-item has a zero value. You can use this clause only with a numeric or numeric-edited screen entry. When you specify this clause, an entry is regarded as numeric edited. When you have an elementary input field, the system ignores this clause.

The BLINK clause causes a literal or data-item field to blink. BLINK has no effect on input fields.

The JUSTIFIED clause specifies right justification of data displayed in a field. Use this clause only with an unedited alphabetic or alphanumeric data-item. JUST is an abbreviation for JUSTIFIED. (This clause is effective only with code revision 9 or greater.)

The BRIGHT and HIGHLIGHT clauses tell the system to use high intensity when it displays the screen item field. The DIM and LOWLIGHT clauses have the opposite effect. When one of these clauses is specified, the system uses low intensity to display a screen item field. (These clauses are only affective with code revision 9 or greater.)

The REVERSE-VIDEO clause tells the system to switch the terminal's background and foreground colors when it displays a data item field. (This clause is only effective with code revision 9 or greater.)

The UNDERLINE clause specifies that each character displayed in the data field will be underlined. (This clause is only effective with code revision 9 or greater.)

Order of Execution

When you execute a screen I/O statement, the system performs terminal functions for individual entries in the following order, regardless of the order in which they appear in a screen description:

BLANK SCREEN or ERASE LINE
LINE and COLUMN positioning
BLANK LINE or ERASE EOL
BELL
REVERSE-VIDEO
UNDERLINE
BLINK
BRIGHT/HIGHLIGHT or DIM/LOWLIGHT
FOREGROUND-COLOR
BACKGROUND-COLOR
DISPLAY or ACCEPT the literal or data-item.

Data Movement with DISPLAY and ACCEPT

Use the DISPLAY and ACCEPT verbs to refer to data in the Screen Section. The DISPLAY verb moves the values of data-items in the File, Working-Storage, or Linkage sections to screen storage. The ACCEPT verb moves the values of data-items in screen storage to the associated data-items in the File, Working-Storage, or Linkage sections. Always execute a DISPLAY statement before an ACCEPT statement, since ACCEPT does not perform an implicit DISPLAY.

Interactive COBOL performs the following steps when executing a DISPLAY sequence:

1. Values move from the File, Working-Storage, or Linkage sections to screen storage:

   1.1. If a Screen Section data-item has a FROM or USING clause, the values of the associated data-item move to screen storage.

   1.2. If a Screen Section data-item has a TO clause, underscores move to screen storage.

   1.3. If a Screen Section item is a literal, its value moves to screen storage.

2. After all data movement is complete, the entire screen is displayed. Data-items with a FROM clause and literals display at half intensity. Data-items with a USING clause display at full intensity. Underscores display at input fields (TO clause).

3. After a DISPLAY statement is executed, the cursor remains after the last displayed field.

When the ACCEPT statement executes:

1. The cursor moves to the first input or update field (described in the Screen Section with a TO or USING clause) and remains there until you complete the field by:

   1.1. Pressing New Line or CR

   1.2. Filling the field (if you specified AUTO)

   1.3. Pressing Esc or a function key

2. If you press New Line, CR, a function key, or fill the field (AUTO), the system validates the entered data against the PICTURE of the associated Screen Section data-item:

   2.1. If the format of the data does not match the PICTURE clause, the system displays an error message. The operator must correct the field before moving to the next input field.

   2.2. If the data in the input or update field is valid, its values move to screen storage. The underscore acts as a terminator. The system ignores the underscore and all characters to the right. The data is redisplayed.

3. If you press New Line or CR, the cursor moves to the next input or update field, and the process is repeated.

4. If you press a function key, the cursor does not advance to the remaining input or update fields.
5. If you press the Escape key, the data in the display field is not validated or moved to screen storage. The cursor does not advance to the remaining input or update fields.
6. After you press the Escape key or a function key, or complete the last input or update field, values of all data-items in the screen referred to by the ACCEPT statement move from screen storage to the corresponding TO or USING items in the File, Working-Storage, or Linkage sections. If you terminate a field by pressing the Escape key, the data in the display field does not move to screen storage, but the value in screen storage is still moved to the data-item in the File, Working-Storage, or Linkage section.
7. If you press the Escape key or a function key, the ON ESCAPE *imper-stmt*, if present, will execute.

Example of Interactive Screen Management

The following example shows the Working-Storage and Screen sections of a program that displays a personnel record. Figure 3–1 shows the screen as it appears on the terminal. The discussion following Figure 3–1 describes screen and operator actions.

```
WORKING-STORAGE SECTION.

01  PERSONNEL-REC.
    05  EMP-NAME         PIC X(30).
    05  ADDRESS.
        10  STREET       PIC X(30).
        10  APT-NO       PIC X(5).
        10  CITY         PIC X(24)   VALUE SPACES.
        10  STATE        PIC X(10).
        10  ZIP          PIC 9(5)    VALUE ZEROS.
    05  PHONES.
        10  HOME-PH      PIC X(12).
        10  BUS-PH       PIC X(12).
    05  DEPARTMENT       PIC X(30).
    05  SUPER            PIC X(30).
    05  BADGE            PIC 9(10).

77  DEFAULT-STATE        PIC X(10)   VALUE "N.C.".

SCREEN SECTION.

01  PERSONNEL-SCREEN
    BACKGROUND-COLOR IS WHITE
    FOREGROUND-COLOR IS BLUE.
    05  BLANK SCREEN LINE 1 COLUMN 25 VALUE "PERSONNEL RECORD"
    BLINK UNDERLINE.
    05  LINE 3       VALUE "NAME: ".
    05               PIC X(30) TO EMP-NAME REQUIRED.
```

```
05  LINE 5     VALUE "ADDRESS".
05  LINE 7     COLUMN 5  VALUE "STREET: ".
05             PIC X(30) TO STREET.
05             COLUMN PLUS 2  VALUE "APT NO.: ".
05             PIC X(5) TO APT-NO.
05  LINE 9     COLUMN 5 VALUE "CITY: ".
05             PIC X(24) USING CITY.
05             COLUMN 35  VALUE "STATE: ".
05             PIC X(10) FROM DEFAULT-STATE TO STATE.
05             COLUMN PLUS 2  VALUE "ZIP: ".
05             PIC 9(5) USING ZIP.
05  LINE 11    COLUMN 5 VALUE "HOME PHONE: ".
05             PIC X(12) TO HOME-PH.
05             COLUMN 31 VALUE "BUSINESS PHONE: ".
05             PIC X(12) TO BUS-PH.
05  LINE 13    VALUE "DEPARTMENT: ".
05             PIC X(30) TO DEPARTMENT REQUIRED.
05  LINE 15    VALUE "SUPERVISOR: ".
05             PIC X(30) TO SUPER.
05  LINE 17    VALUE "BADGE NUMBER: ".
05             PIC 9(10) TO BADGE REQUIRED FULL.
```

```
                    PERSONNEL RECORD

  NAME: ________________________

  ADDRESS

     STREET: _____________________  APT NO.: ____

     CITY: Research Triangle Park    STATE: N.C.   ZIP: 27709

     HOME PHONE: _________  BUSINESS PHONE: _________

  DEPARTMENT: ________________________

  SUPERVISOR: ________________________

  BADGE NUMBER: ________
```

Figure 3-1 Sample Screen Definition

The execution of the statements DISPLAY PERSONNEL-SCREEN and ACCEPT PERSONNEL-SCREEN has the following effects:

1. The program blanks the screen and displays the literal entries, the current values of USING and FROM data-items, and underscores, which define the input entries. All characters, except the CITY and ZIP fields, display at half intensity.
2. The cursor moves to the first position in the field following NAME. Since this is an input field (defined with a TO clause), it is delimited by underscores. As you enter data, the characters display at full intensity. Since EMP-NAME includes the REQUIRED clause, you must type at least one character before terminating the entry. When you press a terminator (except Esc), the data is verified against the PICTURE clause and redisplayed if valid. If you enter invalid data, the cursor moves to the first incorrect character. When the data is correct, press a terminator to move to the next field.
3. The fields following STREET and APT NO. can receive data or remain blank.
4. The fields following CITY and ZIP are update fields (defined with a USING clause) and, as such, retain their current values. In Figure 3-1 the values of CITY and ZIP are displayed. Pressing a terminator reassigns the displayed values to screen storage and moves the cursor to the next field. You can modify displayed data by typing in new data.
5. The field following STATE is defined with both a FROM and a TO clause. This means that N.C., the value of DEFAULT-STATE, is reassigned to the field each time a DISPLAY is executed. As with CITY and ZIP, pressing a terminator assigns the current value of DEFAULT-STATE to screen storage. You can modify displayed data by typing in new data.
6. The fields following HOME PHONE and BUSINESS PHONE can receive data or remain blank.

7. You must enter at least one character for DEPARTMENT. After you enter the data and press a terminator, the field is redisplayed.
8. The field following SUPERVISOR can receive data or remain blank.
9. The field following BADGE NUMBER must be filled. If you enter too few characters, an error message is displayed, and the cursor moves to the end of the entered data and waits for the field to be filled.
10. When the field is full and you press a terminator, the values of all data-items in the Screen Section referenced by the ACCEPT statement move to the corresponding TO or USING items in the File, Working-Storage, and Linkage sections.
11. Execution of the ACCEPT statement is complete. Control passes to the next executable statement in the program.

End of Chapter

Chapter 4
COBOL Source Entry

You can use any text editor for writing COBOL source text. The source file can be in CRT or card format.

CRT Format

In CRT format, the only restrictions imposed are the contents of areas A and B and the use of indicator characters. The compiler reads characters until it finds a line terminator or until it has read 132 characters, whichever comes first.

A line of Interactive COBOL source code in CRT format has the following layout:

```
Indicator  |    Area     |    Area
  area     |     A       |     B
(optional) |             |
```

Indicator area (optional) — Column 1. Only an indicator character (d, D, *, –, or /) is permitted in this one-character optional field.

Area A — Columns 2–5 if the line has an indicator character, and columns 1–4 without an indicator. The following COBOL elements must begin in area A: division headers, section-names, FD entries, paragraph-names, data descriptions at the 01 and 77 levels, and the phrases DECLARATIVES and END DECLARATIVES. A tab in area A terminates the area, with area B beginning in the next column.

Area B — Columns 6–132 if the line has an indicator character, and columns 5–132 without an indicator. Continuation lines, sentences, and statements must begin in area B. A tab in area B is equivalent to a space.

Comment lines and level numbers 02–49 and 88 can begin in area A or B. Figure 4–1 gives an example of COBOL source code in CRT format.

```
SCREEN SECTION.
    COPY "SCREEN10".
01  MASTER-MENU.
    02  LINE 2 BLANK SCREEN COLUMN 20 VALUE "THE FUNCTIONS
-       " AVAILABLE ARE: ".
    02  LINE PLUS 2 COLUMN 20 VALUE "1) ADD RECORD".
    02  LINE PLUS 1 COLUMN 20 VALUE "2) CHANGE RECORD".
    02  LINE PLUS 1 COLUMN 20 VALUE "3) EXAMINE RECORD".
    02  LINE PLUS 1 COLUMN 20 VALUE "4) DELETE RECORD".
    02  LINE PLUS 2 COLUMN 10 BELL VALUE "FUNCTION? ".
    02  COLUMN PLUS 1 PIC 9 TO CHOICE.


    COPY "SCREEN15".
```

Figure 4–1 COBOL Source Code in CRT Format

Card Format

A line of Interactive COBOL source code in card format has the following layout:

Sequence number area	Indicator area	Area A	Area B	Comment area

Sequence number area

Columns 1-6. This 6-character field typically contains a line number, and must be present in all source lines in the main program and in all COPY files.

The compiler checks only for the presence of this field, and does not check its contents. Therefore, it can contain letters, tabs, nonsequential numbers, spaces, and so forth.

If you do not want numbers, letters, or spaces in columns 1–6, enter a tab. A tab in the sequence number area terminates the area.

Indicator area

Column 7. If the sequence number area contains a tab, the system inspects the next character. If it is an indicator character (d, D, *, –, or /), column 7 is considered an indicator area.

If the sequence number area does not contain a tab, column 7 must contain a space or an indicator character.

Area A

Columns 8–11 if the line has an indicator character, and columns 7–10 without an indicator. The following COBOL elements must begin in area A: division headers, section-names, paragraph-names, FD entries, data descriptions at the 01 and 77

	levels, and the phrases DECLARATIVES and END DECLARATIVES.
	A tab in area A terminates the area, with area B beginning in the next column.
Area B	Columns 13–72. Continuation lines, sentences, and statements must begin in area B. A tab in area B is equivalent to a space.
Comment Area	Columns 73–80. Characters in the comment area are placed in the compilation listing but do not affect program compilation or execution.

Comment lines and level numbers 02–49 and 88 may begin in area A or B. Figure 4–2 gives an example of COBOL source code in card format.

```
001000 SCREEN SECTION.
001010
001020      COPY "SCREEN10".
001030
001040 01   MASTER-MENU.
001050      02 LINE 2 BLANK SCREEN COLUMN 20 VALUE "THE
001060-         " FUNCTIONS AVAILABLE ARE: ".
001070      02 LINE PLUS 2 COLUMN 20 VALUE "1) ADD RECORD".
001080      02 LINE PLUS 1 COLUMN 20 VALUE "2) CHANGE RECORD".
001090      02 LINE PLUS 1 COLUMN 20 VALUE "3) EXAMINE RECORD".
002000      02 LINE PLUS 1 COLUMN 20 VALUE "4) DELETE RECORD".
002010      02 LINE PLUS 2 COLUMN 10 BELL VALUE "FUNCTION? ".
002020      02 COLUMN PLUS 1 PIC 9 TO CHOICE.
002030
002040      COPY "SCREEN15".
```

Figure 4–2 COBOL Source Code in CARD Format

Continuation Lines

The continuation indicator (–) breaks a word or literal between successive lines. When the continuation indicator is present, the compiler ignores the terminator of the previous line and interprets the text on the current line as a continuation. The following rules apply to continued lines:

- The text on the continued line must begin in area B.
- When a COBOL word is broken between two lines, the first nonblank character of the continuation line follows the last character of the previous line. For example:

  line 1: <Tab>DIVIDE GROSS-SALES BY TOT-NUM-SALES GIV<NL>

  line 2: –<Tab>ING AVERAGE-SALES; ON SIZE ERROR PERFORM

  The compiler interprets the continued word as GIVING.

- When a continued statement, phrase, or entry is broken between two words, a continuation indicator is not necessary. If a hyphen is not present in the indicator area of a line, the compiler assumes that the last character on the previous line is followed by a space. For example:

  ```
  line 1: 004210 DIVIDE GROSS-AMT BY TOTAL-SALES<NL>
  line 2: 004220        GIVING AVERAGE-SALES.
  ```

  The compiler interprets these lines as:

  ```
  DIVIDE GROSS-AMT BY TOTAL-SALES GIVING AVERAGE-SALES.
  ```

- When a nonnumeric literal is broken between two lines, the continued line must begin with a quotation mark. In this case, the space following the word REPORT appears between the words REPORT and OF when the literal is output:

  ```
  line 1:  DISPLAY "THIS IS THE ANNUAL REPORT <NL>
  line 2: -     "OF THE CENTRAL HARDWARE CORP.".
  ```

  When the statement executes, the runtime system displays:

  ```
  THIS IS THE ANNUAL REPORT OF THE CENTRAL HARDWARE CORP.
  ```

Comment Lines

You can enter comments anywhere in the source code, and a program can contain multiple comment lines. Indicate a comment line by typing an asterisk (*) or slash (/) in the indicator area. When you use the asterisk, the compiler advances one line before writing to the output listing. When you use the slash, the compiler advances to a new page. The text in a comment line is produced in the listing but has no effect on compilation or execution.

Debugging Lines

With Interactive COBOL revision 1.60 or greater, you can include debugging lines anywhere in your program by placing a d or D in the indicator area of your program. Then, if you use the appropriate compiler switch to compile the program, these lines are compiled. Otherwise, Interactive COBOL treats them as comment lines. These lines are also compiled when you include the WITH DEBUGGING MODE clause in the SOURCE–COMPUTER paragraph.

See your user's guide for information on compiler switches.

Blank Lines

You can insert blank lines in the source code at any point, except immediately before a line with a continuation indicator.

End of Chapter

Chapter 5
The Identification Division

The Identification Division must be the first division of a source program. It consists of the required PROGRAM-ID paragraph and several optional paragraphs that can be included for documentation.

The general format of the Identification Division is:

IDENTIFICATION DIVISION.

PROGRAM-ID. *program-name*.
[AUTHOR. [*comment entry*] ...].
[INSTALLATION. [*comment entry*]...].
[DATE-WRITTEN. [*comment entry*]...].
[DATE-COMPILED. [*comment entry*]...].
[SECURITY. [*comment entry*]...].

The PROGRAM-ID paragraph must be the first paragraph in the Identification Division. The program-name is required and can be any valid COBOL data-name, but it is used only for documentation. It is not the name by which the program is known to the system. The external filename of the file containing the source code is the name used to compile and execute the program.

The optional paragraphs must be used in the order shown. A comment entry can be one or more lines long and can appear in area A or B.

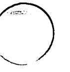
The DATE-COMPILED paragraph does not affect compilation. The system date is placed in the compilation listing on the line following the DATE-COMPILED entry.

The following is an example of the Identification Division:

```
IDENTIFICATION DIVISION.

PROGRAM-ID.          INVENTORY-01.
AUTHOR.              HEADQUARTERS PROGRAMMING GROUP.
INSTALLATION.        CENTRAL HARDWARE DISTRIBUTORS INC.
DATE-WRITTEN.        JANUARY, 1990.
DATE-COMPILED.       JANUARY, 1990.
SECURITY.            NOT APPLICABLE.
```

End of Chapter



Chapter 6
The Environment Division

The Environment Division provides a standard way of expressing program values that are dependent on the physical characteristics of a specific computer. It must follow the Identification Division and consists of two sections: the Configuration Section and the Input-Output Section.

The general format of the Environment Division is:

```
[ ENVIRONMENT  DIVISION.
[ CONFIGURATION  SECTION.

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

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

[ SPECIAL-NAMES.  [ SWITCH literal
    [; IS mnemonic-name]
    [;  ON STATUS  IS condition-name]
    [;  OFF STATUS  IS condition-name]]...
    [                    { NATIVE     } ]
    [; alphabet-name IS  { STANDARD-1 } ]
    [;  CURRENCY SIGN IS lit]
    [;  DECIMAL-POINT IS  COMMA ].]]

[ INPUT-OUTPUT  SECTION.

FILE-CONTROL. file-control-entry...
[ I-O-CONTROL. [i-o-control-entry]...].]]
```

The following is an example of the Environment Division:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. MV-15000.
OBJECT-COMPUTER. DASHER-386.
SPECIAL-NAMES.
    SWITCH "A", ON STATUS IS SW0, OFF STATUS IS SW1
    SWITCH "B", OFF STATUS IS SW2, ON STATUS IS SW3
    CURRENCY SIGN IS "M"
    DECIMAL-POINT IS COMMA.
```

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT AFILE ASSIGN TO DISK.
    SELECT BFILE ASSIGN TO DISK.
I-O-CONTROL.
    SAME RECORD AREA FOR AFILE, BFILE.
```

Configuration Section

The Configuration Section names the computers that compile and execute the program. It also defines switches and other special conditions.

The SOURCE-COMPUTER paragraph names the computer used to compile the program; the OBJECT-COMPUTER paragraph names the computer used to run the program. Since both paragraphs are used for documentation only, you can leave them empty. The computer name can be any valid COBOL data-name.

The WITH DEBUGGING MODE clause tells the compiler to compile the debugging lines when you are using Interactive COBOL revision 1.60 or greater. Without this clause, the compiler treats the debugging lines as comment lines (unless you include the appropriate switch on your compiler command line).

MEMORY SIZE can be any integer value. This clause and the PROGRAM COLLATING SEQUENCE clause are used for documentation only.

In the SPECIAL-NAMES paragraph, the SWITCH literal must be an uppercase or lowercase letter from A to Z. This literal names a logical switch that can be tested for an ON or OFF condition. The ON and OFF clauses can appear in either order. At least one of the clauses must be present when a switch is named. A switch value is set at runtime and remains unchanged while the program is running. The condition-name describing the ON or OFF status need not be defined elsewhere; no Working-Storage entry is needed. The SWITCH *literal* IS *mnemonic-name* clause is used for documentation only.

Alphabet-name is a programmer-defined data name that refers to a specific character set or collating sequence. The *alphabet-name* IS clause is used for documentation only.

The literal in the CURRENCY SIGN clause represents the currency symbol in the PICTURE clause. The literal is a single character, but it *cannot* be one of the following:

Digits 0–9
Letters A B C D P R S V X Z
Space character
Special characters * + – ( ) / = , . " ;

If you omit this clause, the currency symbol defaults to the dollar sign ($).

NOTE: On revisions of Interactive COBOL prior to 1.60, you cannot use the letter L as the literal in the CURRENCY SIGN clause.

The DECIMAL-POINT IS COMMA clause exchanges the functions of the period and comma in numeric literals and in the PICTURE clause character string. All

occurrences of commas in numeric literals are changed when you use this clause. If your source code contains an index or subscript entry—for example, (4, 2)—the comma changes to a period and the compiler generates an error.

Input-Output Section

The Input-Output Section consists of two paragraphs that supply information needed to control the transmission and manipulation of data between the object program and external devices. These two paragraphs are called FILE-CONTROL and I-O-CONTROL.

The FILE-CONTROL Paragraph

An entry in the FILE-CONTROL paragraph specifies the name of each file used by the program, associates the file with a particular hardware device, and designates other file characteristics.

The SELECT Clause

The SELECT clause names internal program files and associates each file with a given hardware device and external filename. Use the SELECT clause to specify logical file organization, access method, file-status items, keys, and file size.

The SELECT formats for sequential, relative, and indexed file organizations are:

Sequential SELECT

```
                                  { PRINTER   }
                                  { PRINTER-1 }
SELECT filename  ASSIGN TO        { DISPLAY   }  [; id-lit]
                                  { KEYBOARD  }
                                  { DISK      }

  [; ORGANIZATION IS SEQUENTIAL ]
  [; ACCESS MODE IS SEQUENTIAL ]
  [;  FILE STATUS IS data-name]
  [; DATA SIZE IS integer].
```

Relative SELECT

```
SELECT filename ASSIGN TO DISK [; id-lit]

    ; ORGANIZATION IS RELATIVE

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

[; FILE STATUS IS data-name]
[; INDEX SIZE IS integer]
[; DATA SIZE IS integer].
```

Indexed SELECT

```
SELECT filename ASSIGN TO DISK [; id-lit]
    ; ORGANIZATION IS INDEXED
    [ ; ACCESS MODE IS { SEQUENTIAL } ]
    [                  { RANDOM     } ]
    [                  { DYNAMIC    } ]
    ; RECORD KEY IS data-name
    [; ALTERNATE RECORD KEY IS data-name [WITH DUPLICATES ] ] ...
    [; FILE STATUS IS data-name]
    [; INDEX SIZE IS integer]
    [; DATA SIZE IS integer].
```

The following is an example of a SELECT clause for a single-keyed indexed file:

```
SELECT CFILE ASSIGN TO DISK;
    ORGANIZATION IS INDEXED;
    ACCESS MODE IS DYNAMIC;
    RECORD KEY IS C-KEY;
    FILE STATUS IS CFSTAT.
```

Fixed and variable sequential files can be assigned only to DISK, and line sequential files can be assigned to PRINTER, PRINTER-1, DISPLAY, or KEYBOARD. The records can only be accessed sequentially.

Relative and indexed files can be assigned only to DISK. If you do not specify access mode, sequential is assumed. In relative and indexed files, sequential access is in ascending order of the RELATIVE KEY or RECORD KEY. In random access, the value of the RELATIVE KEY or RECORD KEY determines the record to be accessed. If you specify dynamic access mode, access can be sequential or random.

Table 6–1 lists the access mode and assignment statement of the different file types. A line sequential file assigned to PRINTER or DISPLAY must be opened for OUTPUT or EXTEND; one assigned to KEYBOARD must be opened for INPUT.

Table 6-1 Interactive COBOL File Handling

File Type	Access Mode: Sequential	Random	Dynamic	Assign to: Printer	Display	Keyboard	Disk
Fixed Seq.	yes	no	no	no	no	no	yes
Variable Seq.	yes	no	no	no	no	no	yes
Line Seq.	yes	no	no	yes	yes	yes	no
Relative	yes	yes	yes	no	no	no	yes
Indexed	yes	yes	yes	no	no	no	yes

Assign a line sequential file to DISPLAY to bypass the Interactive COBOL Screen Section and send output directly to the terminal screen. To read a line sequential disk file, assign to KEYBOARD. See Table 2-2 in Chapter 2 for line terminator information.

To create a line sequential file on disk, assign to DISPLAY or PRINTER, and specify an external filename. For example:

```
SELECT CUST-RPT ASSIGN TO PRINTER, "CUST$RPT".
```

The id-lit following the file device type in the SELECT clause is an Interactive COBOL extension. It allows a program to associate an internal filename with an external filename.

The ORGANIZATION clause specifies the logical structure of a file. File organization is established when the file is created and can be changed only by the REORG utility. The three types of file organization are sequential, indexed, and relative.

The ACCESS MODE clause designates the method used in retrieving records from a file or writing records to a file. The three access modes are random, dynamic, and sequential.

The RECORD KEY and ALTERNATE RECORD KEY data-names must be defined as alphanumeric items within a record description entry associated with the filename. Each record key can have a maximum length of 100 characters and can be qualified.

The data-item in the RECORD KEY clause is the primary record key for the file and must be unique among records of the file. This key provides an access path to records in an indexed file.

Each data-item in the ALTERNATE RECORD KEY clause is an alternate record key for the file. An alternate key provides an additional access path to records in an indexed file. You can specify up to four ALTERNATE RECORD KEY clauses.

When a data description refers to an existing file, the descriptions of the RECORD KEY and the ALTERNATE RECORD KEY data-names, as well as their relative locations within the data record, must be the same as when the file was created.

The DUPLICATES phrase specifies that the value of the associated alternate record key can be duplicated within any of the records of the file. This phrase is used for documentation only. Interactive COBOL allows duplicates even if the DUPLICATES phrase is not specified.

The RELATIVE KEY phrase specifies a key whose contents identify a logical record in a relative file. Relative keys must be defined as PIC 9(4) COMP to PIC 9(18) COMP. See the "Relative Files" section in Chapter 2 for more information.

The FILE STATUS data-item holds a value that indicates the outcome of an I/O operation. It must be described as a two-character alphanumeric item. See your user's guide for a list of File Status codes.

The INDEX SIZE clause and DATA SIZE clause are Interactive COBOL extensions. These clauses specify the size of the file elements for the index and data portions of an indexed or relative file. On AOS/VS, storage space for files is dynamically allocated. The storage space is allocated in sets of contiguous 512-byte blocks called file elements. The INDEX SIZE and DATA SIZE clauses specify the size of the file elements for the indexed and data portions of an indexed or relative file. (AOS/VS ignores the DATA SIZE clause if it refers to a sequential file.) The size of each clause must be a multiple of 4. If the supplied value is not a multiple of 4, it is rounded up to the next higher multiple of the AOS/VS default element size. If these clauses are not present, the AOS/VS default file element size is used.

The INDEX SIZE and DATA SIZE clauses are ignored under AOS (only), MS-DOS, and UNIX systems. See your user's guide for more information if you are using the DG/RDOS operating system.

The I-O-CONTROL Paragraph

The format for an I-O-CONTROL entry is:

[SAME [RECORD] AREA FOR *filename*, {*filename*}...]

The following is an example:

```
I-O-CONTROL.
    SAME RECORD AREA FOR GOOD-CUST-FILE, CREDIT-RISK-FILE
    SAME RECORD AREA FOR PARTS-1, PARTS-2, PARTS-3.
```

The SAME AREA and SAME RECORD AREA clauses are treated identically. In both cases, the specified files share a common record area. More than one of the files can be open at any given time. Since the two clauses are treated in the same manner, it is illegal for a filename to appear in more than one of the clauses.

End of Chapter

Chapter 7
The Data Division

The Data Division describes the data that the object program creates, manipulates, uses as input, or produces as output. It must follow the Environment Division in the source program.

The Data Division consists of four sections: the File Section, the Working-Storage Section, the Linkage Section, and the Screen Section, which is an Interactive COBOL feature.

The general format of the Data Division is:

[DATA DIVISION.

[FILE SECTION. [FD-*entry*
  record-description-entry...]...]

[WORKING-STORAGE SECTION.
  [*data-description-entry*]...]

[LINKAGE SECTION.
  [*data-description-entry*]...]

[SCREEN SECTION.
  [*data-description-entry*]...]]

The following example shows the Data Division of an Interactive COBOL program:

```
DATA DIVISION.

FILE SECTION.
FD  F-CUSTOMER LABEL RECORD IS OMITTED
    DATA RECORD IS F-CUST-REC.
01  F-CUST-REC.
    03  FAREA       PIC 99.
    ...
    03  FADDRESS.
        05 FSTREET PIC X(20).
        ...
WORKING-STORAGE SECTION.
01  FUNCTION        PIC 99 VALUE 00.
77  WORDAMT         PIC 9(5)V99 VALUE ZERO.
01  SECURITY-CODE.
    03  W-CATEGORY      PIC X VALUE " ".
   ...
```

```
SCREEN SECTION.
01  ACCEPT-SECURITY.
    03  LINE 1 BLANK SCREEN.
    03  LINE PLUS 1 COLUMN 10 VALUE "PLEASE ENTER AUTHORIZATION".
    03  S-CATEGORY COLUMN 50 PIC X TO W-CATEGORY SECURE AUTO.
```

The File Section

The File Section consists of FD entries. These entries contain record descriptions that Interactive COBOL uses to transfer data to and from files.

FD Entry

The FD entry describes the physical attributes of a file. It contains the file declaration that the SELECT clause began. You must specify one FD entry for each file declared in a SELECT clause. You must follow each FD entry with one or more 01-level record description entries. The general format for the FD entry is:

```
FD filename
[ ; BLOCK CONTAINS integer { RECORDS    } ]
                           { CHARACTERS }
[; RECORD CONTAINS integer [ TO integer] CHARACTERS ]
[ ; RECORDING MODE IS { VARIABLE } ]
                      { FIXED    }
; LABEL { RECORD IS   } { STANDARD }
        { RECORDS ARE } { OMITTED  }
[ ; DATA { RECORD IS   } { data-name}... ]
         { RECORDS ARE }
[; CODE-SET IS alphabet-name].
```

The BLOCK CONTAINS clause is used for documentation only.

The RECORD CONTAINS clause defines the record size or the record size limits of the file. The maximum record length for files assigned to PRINTER, DISPLAY, or KEYBOARD is 254 bytes for UNIX, AOS/VS, and MS-DOS systems, and 132 bytes for AOS and DG/RDOS systems. The maximum record length for indexed or relative files is 4096 bytes. For a sequential file assigned to DISK, the resulting program file cannot exceed the maximum program size for your system.

The RECORDING MODE clause, which is an extension to ANSI COBOL, is permitted only for fixed and variable sequential files. It indicates whether the file contains fixed or variable length records. If you omit this clause, fixed length is assumed.

The LABEL clause is required, even though it is used for documentation only.

The DATA RECORD and CODE-SET clauses are used for documentation only.

Table 7-1 lists the clauses that are legal in the Data Division sections.

Table 7-1 Legal Entries in the Four Data Division Sections

Clause or Entry	File	W-S	Linkage	Screen
FD entry	Y	N	N	N
BLOCK CONTAINS	Y	N	N	N
RECORD CONTAINS	Y	N	N	N
RECORDING MODE	Y	N	N	N
LABEL RECORDS	Y	N	N	N
DATA RECORDS	Y	N	N	N
CODE-SET	Y	N	N	N
FILLER	Y	Y	Y	N
REDEFINES	Y	Y	Y	N
PICTURE	Y	Y	Y	Y
USAGE	Y	Y	Y	N
SIGN	Y	Y	Y	N
OCCURS	Y	Y	Y	N
SYNCHRONIZED	Y	Y	Y	N
JUSTIFIED	Y	Y	Y	Y
BLANK WHEN ZERO	Y	Y	Y	Y
VALUE	N	Y	N	Y
Level 77	N	Y	Y	N
Level 88	Y	Y	Y	N
AUTO	N	N	N	Y
SECURE	N	N	N	Y
REQUIRED	N	N	N	Y
FULL	N	N	N	Y
FROM	N	N	N	Y
TO	N	N	N	Y
USING	N	N	N	Y
BELL	N	N	N	Y
BLINK	N	N	N	Y
BLANK SCREEN	N	N	N	Y
LINE	N	N	N	Y
COLUMN	N	N	N	Y
BLANK LINE	N	N	N	Y

Working-Storage Section

The Working-Storage Section must follow the File Section. It describes records and elementary data-items that are not part of external files, but are developed and processed internally. These elements function as temporary data storage items during program execution.

Linkage Section

The Linkage Section appears only in a program that is called by another program. The data descriptions in the Linkage Section enable a called program to reference data from the calling program.

You must define data-items that are passed in either the File, Working-Storage, or Linkage sections of the calling program. You must define corresponding data-items that receive the passed information in the Linkage Section of the called program.

The Linkage Section must immediately follow the Working-Storage Section. Its structure is the same as that of the Working-Storage Section.

The size of the Linkage Section depends on the operating system. On DG/RDOS and MS-DOS, the system allocates up to 12 KB of disk space per user. On AOS, AOS/VS, and UNIX systems, the Linkage Section is limited only by program size.

Linkage Section data descriptions can use the same descriptor clauses that appear in the Working-Storage Section, with one exception: you cannot use the VALUE clause in a Linkage Section data entry except with level 88 items.

Each sending item defined in a calling program and each receiving item defined in the Linkage Section of a called program must have a level number of 01 or 77.

Corresponding sending and receiving data-items must have the same total length, but need not have the same identifier or the same internal structure. A list of sending data-items must appear in a CALL USING or CALL PROGRAM USING statement of the calling program (see CALL and CALL PROGRAM in Chapter 8). A list of the corresponding receiving items must appear, in the same order, in the Procedure Division Using header of the called program.

You can use an identifier more than once in a CALL PROGRAM USING statement. You cannot use it more than once in a Procedure Division Using header.

You can define any number of items in the Linkage Section. However, the called program can refer to only items named in the receiving list and items subordinate to them.

The following example shows the relevant entries in a calling program and in the called program.

In the calling program:

```
DATA DIVISION.
...
WORKING-STORAGE SECTION.
...
01  SENDING-RECORD      PIC X(25).
01  PHONE-NUMBER.
    03  AREA-CODE       PIC 999.
    03  EXCHANGE        PIC 999.
    03  BASE-NUMBER     PIC 9(4).
77  TODAYS-DATE         PIC 9(6).
```

In the called program:

```
DATA DIVISION.
...
LINKAGE SECTION.
77  RECEIVING-RECORD    PIC A(25).
```


Licensed Material - Property of Data General Corporation

```
01  PHONE-NUMBER.
    03  AREA-CODE       PIC 999.
    03  EXCHANGE        PIC 999.
    03  BASE-NUMBER     PIC 9(4).
01  TODAYS-DATE.
    03  T-MONTH     PIC 99.
    03  T-DAY       PIC 99.
    03  T-YEAR      PIC 99.
```

When the calling program executes the following statement, the system preserves the values of the listed data-items:

```
CALL PROGRAM "GET" USING SENDING-RECORD,
    PHONE-NUMBER, TODAYS-DATE.
```

When the called program (GET) begins execution, the procedure header

```
PROCEDURE DIVISION USING RECEIVING-RECORD,
         PHONE-NUMBER, TODAYS-DATE.
```

causes the system to initialize the listed items to the values of the preserved items from the calling program.

Screen Section

The Screen Section defines the attributes of screens used in interactive screen I/O. The Screen Section data descriptions are similar in format to other sections of the Data Division. However, additional screen clauses allow you to specify the position of items on the display screen and use other console controlling functions. You reference a Screen Section entry in the Procedure Division with the DISPLAY and ACCEPT verbs.

Screen Data Description Entry

A Screen Section can describe a literal or a data-item entry. A literal entry displays constant information such as a title or a menu. A data-item entry is considered a "window" through which data is transferred from the screen to program storage, from program storage to the screen, or in both directions. Each data-item entry must contain a PICTURE clause, and a FROM, TO or USING clause to identify an associated File, Working-Storage, or Linkage Section item.

You can combine literal and data-item entries to form a group entry. A group entry is defined with level numbers in a format similar to that of group items in the File and Working-Storage sections. The following figures show the syntax for describing screen entries.

Literal Format

```
level-number  [screen-name]
     [; BLANK SCREEN ]
     [; BLANK LINE ]
     [; BELL ]
     [; BLINK ]
     [; ERASE EOL ]
     [; ERASE EOS ]
     [; ERASE LINE ]
     [; REVERSE-VIDEO ]
     [; UNDERLINE ]

     [  { LOWLIGHT  } ]
     [ ;{ HIGHLIGHT } ]
     [  { DIM       } ]
     [  { BRIGHT    } ]

     [                    { integer-1         } ]
     [ ; LINE NUMBER IS   { PLUS integer-2    } ]
     [                    { identifier-1      } ]

     [  { COLUMN }             { integer-1      } ]
     [ ;{        } NUMBER IS   { PLUS integer-2 } ]
     [  { COL    }             { identifier-1   } ]

     [                             { integer-3 } ]
     [                             { BLACK     } ]
     [                             { BLUE      } ]
     [                             { GREEN     } ]
     [ ; FOREGROUND-COLOR IS       { CYAN      } ]
     [                             { RED       } ]
     [                             { MAGENTA   } ]
     [                             { BROWN     } ]
     [                             { WHITE     } ]

     [                             { integer-4 } ]
     [                             { BLACK     } ]
     [                             { BLUE      } ]
     [                             { GREEN     } ]
     [ ; BACKGROUND-COLOR IS       { CYAN      } ]
     [                             { RED       } ]
     [                             { MAGENTA   } ]
     [                             { BROWN     } ]
     [                             { WHITE     } ]

     [; VALUE IS literal].
```

Data-Item Format

```
level-number  [screen-name]
    [; BLANK  SCREEN ]
    [; BLANK  LINE ]
    [; BELL ]
    [; BLINK ]
    [; ERASE EOL ]
    [; ERASE EOS ]
    [; ERASE LINE ]
    [; REVERSE-VIDEO ]
    [; UNDERLINE ]
    [; { LOWLIGHT  } ]
       { HIGHLIGHT }
       { DIM       }
       { BRIGHT    }
    [; LINE NUMBER IS { integer-1         } ]
                      { PLUS integer-2    }
                      { identifier-1      }
    [; { COLUMN } NUMBER IS { integer-1      } ]
       { COL    }           { PLUS integer-2 }
                            { identifier-1   }
    [; FOREGROUND-COLOR IS { integer-3 } ]
                           { BLACK     }
                           { BLUE      }
                           { GREEN     }
                           { CYAN      }
                           { RED       }
                           { MAGENTA   }
                           { BROWN     }
                           { WHITE     }
    [; BACKGROUND-COLOR IS { integer-4 } ]
                           { BLACK     }
                           { BLUE      }
                           { GREEN     }
                           { CYAN      }
                           { RED       }
                           { MAGENTA   }
                           { BROWN     }
                           { WHITE     }
    [; { PICTURE } IS picture-string { ; TO id [ FROM id-lit ] } ]
       { PIC     }                   { ; USING id              }
                                     { ; FROM id-lit [ TO id ] }
    [; BLANK WHEN  ZERO ]     [; { JUSTIFIED } RIGHT]
                                 { JUST      }
    [; AUTO ]
    [; SECURE ]
    [; REQUIRED ]
    [; FULL ].
```

Group Format

```
level-number [{screen-name}]
             [{FILLER     }]

             [; BLANK SCREEN ]
             [; FULL ]
             [; AUTO ]
             [; BLINK ]
             [; SECURE ]
             [; REQUIRED ]
             [; REVERSE-VIDEO ]
             [; UNDERLINE ]

             [ ; { LOWLIGHT  } ]
             [   { HIGHLIGHT } ]
             [   { DIM       } ]
             [   { BRIGHT    } ]

             [                           { integer-1 } ]
             [                           { BLACK     } ]
             [                           { BLUE      } ]
             [                           { GREEN     } ]
             [ ; FOREGROUND-COLOR IS     { CYAN      } ]
             [                           { RED       } ]
             [                           { MAGENTA   } ]
             [                           { BROWN     } ]
             [                           { WHITE     } ]

             [                           { integer-2 } ]
             [                           { BLACK     } ]
             [                           { BLUE      } ]
             [                           { GREEN     } ]
             [ ; BACKGROUND-COLOR IS     { CYAN      } ]
             [                           { RED       } ]
             [                           { MAGENTA   } ]
             [                           { BROWN     } ]
             [                           { WHITE     } ]

             [; [USAGE IS ] DISPLAY]

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

             { data-item     }
             { literal entry }.
```

For examples of complete screen descriptions, see the sample program in Chapter 3.

Level Number

The data description for a Screen Section entry must begin with a level number. Level 01 entries must begin in area A; you can indent other levels as desired. Level numbers in the Screen Section can range from 01 to 49.

Screen-Name

The screen-name must immediately follow the level number. Screen-names must conform to the standard rules for programmer-defined words. Omitting the screen-name from an entry is equivalent to coding a FILLER entry for a Working-Storage data-item. Unnamed items in a group entry cannot be referenced separately. Screen-names are required at the 01 level.

Screen Clauses

You can specify multiple clauses for each entry, and code them in any order. Regardless of their order, however, the clauses execute in the following order:

BLANK SCREEN or ERASE LINE
LINE and COLUMN positioning
BLANK LINE or ERASE EOL
BELL
REVERSE-VIDEO
UNDERLINE
BLINK
HIGHLIGHT/BRIGHT or LOWLIGHT/DIM
FOREGROUND-COLOR
BACKGROUND-COLOR
DISPLAY or ACCEPT the literal or data-item.

Table 7-2 summarizes the Screen Section clauses (see Chapter 3 for more information).

Table 7-2 Screen Section Clauses

Clause	Meaning
AUTO	Terminates an input or update field automatically when the field is full.
BACKGROUND-COLOR	Specifies the background color for a screen item when you are using code revision 9 or greater.
BELL	Sounds the console's tone.
BLANK LINE	Erases a line from the current cursor position to the end of the line and leaves the cursor position unchanged.
BLANK SCREEN	Erases the screen and positions the cursor to line 1, column 1.
BLANK WHEN ZERO	Substitutes spaces for a numeric or numeric edited item when its value is zero.
BLINK	Causes a field to blink.
BRIGHT	Causes the field to be displayed using high intensity when you are using code revision 9 or greater.
COLUMN	Specifies horizontal positioning.
DIM	Causes the field to be displayed using low intensity when you are using code revision 9 or greater.
ERASE EOL	Erases the screen from the current cursor position to the end of the line and leaves the cursor position unchanged when you are using code revision 9 or greater.
ERASE EOS	Erases the screen position from the current cursor position through the end of the screen when you are using code revision 9 or greater.
ERASE LINE	Erases an entire line when you are using code revision 9 or greater.
FOREGROUND-COLOR	Specifies the foreground color for a screen item when you are using code revision 9 or greater.
FROM	Identifies an output field.
FULL	Requires a character or space entry in each position. Used with input or update fields.
HIGHLIGHT	Causes the field to be displayed using high intensity when you are using code revision 9 or greater.
JUSTIFIED	Functions the same way as in other sections of the Data Division when you are using code revision 9 or greater.
LINE	Specifies vertical positioning.
LOWLIGHT	Causes the field to be displayed using low intensity when you are using code revision 9 or greater.

Table 7-2 Screen Section Clauses *(concluded)*

Clause	Meaning
PICTURE	Describes the type, size, and appearance of a data-item entry and controls the way in which data is accepted or displayed.
REQUIRED	Indicates the need to enter at least one character. Used with input or update fields.
REVERSE-VIDEO	Causes the data field to be displayed with the foreground and background colors reversed (i.e., the foreground color becomes the background color and the background color becomes the foreground color) when you are using code revision 9 or greater.
SECURE	Causes asterisks to echo on the screen during data entry. Used with input or update fields.
SIGN	Specifies that there is a separate sign character and holds that character's position in the field when you are using code revision 9 or greater.
TO	Identifies an input field.
UNDERLINE	Causes each character of the field to be underlined when it is displayed when you are using code revision 9 or greater.
USAGE	Specifies the format of the data item in computer storage when you are using code revision 9 or greater.
USING	Identifies an update field.
VALUE	Specifies literal information to be displayed.

Data Description Entries

The level-number, data-name or FILLER clause, and the REDEFINES clause must appear in the order listed. You can code the remaining clauses in any order. The general format for a Data Description entry is:

```
level-number [ { data-name } ]
               { FILLER    }
   [ ;   REDEFINES data-name]
     ; { PICTURE }  IS picture-string
       { PIC     }
   [ ; [ USAGE IS ] { COMPUTATIONAL } ]
                    { COMP          }
                    { DISPLAY       }
                    { INDEX         }
   [ ; [ SIGN IS ] { LEADING  } [ SEPARATE CHARACTER ] ]
                   { TRAILING }
   [ ; OCCURS integer TIMES [ ; INDEXED BY { index-name }...] ]
   [ ; { SYNCHRONIZED } { LEFT  } ]
       { SYNC         } { RIGHT }
   [ ; { JUSTIFIED } RIGHT ]
       { JUST      }
   [ ; BLANK WHEN ZERO ]
   [ ; VALUE IS literal].
```

Level Numbers

Level numbers must always be the first element in an entry. Level numbers can be 01, 02–49, 77, and 88.

Level 01

Level number 01 identifies the record as a whole. Level 01 can be used in any section of the Data Division and must begin in area A. A level 01 data-item always begins on an even byte boundary with code revision 7.

Multiple 01-level entries subordinate to an FD entry implicitly redefine the same memory area. Data description entries subordinate to an FD entry must have level numbers 01–49 or 88.

Levels 02–49

Levels 02–49 identify the hierarchy of data within the record. Levels 02–49 can be used in any section in the Data Division and must begin in area B.

Level 77

Level number 77 identifies elementary data-items and is assigned to entries where there is no real concept of level. Level 77 items cannot be subdivisions of other items,

and cannot be subdivided. Level 77 items can have subordinate level 88 entries. Level 77 must begin in area A and can be used only in the Working-Storage and Linkage sections. Level 77 always begins on an even byte boundary with code revision 7.

Level 88

Level number 88 defines condition-names. Level 88 items can begin in area A or B and can occur only in the File, Working-Storage and Linkage sections. The condition-name must be followed by a VALUE clause specifying the literal or literals that apply to that condition. The condition-name can then be used in place of a relation condition (see Condition-Name Condition in Chapter 1 for an example of a level 88 format).

Data-Name/FILLER Clause

A data-name entry specifies the name of the data-item being described. A FILLER entry specifies an elementary item in a logical record that cannot be referred to explicitly. The data-name or FILLER clause must immediately follow the level number. Omitting this clause is equivalent to entering FILLER. FILLER cannot be used in the Screen Section after a 01 level number.

REDEFINES Clause

The REDEFINES clause allows you to describe the same memory area in different data description entries. For example:

```
01  TAX-GROUPS.
    03  GROUP-A.
        05  A-ITEM OCCURS 6 TIMES PIC 9999.
    03  GROUP-B REDEFINES GROUP-A.
        05  B-ITEM OCCURS 12 TIMES PIC XX.
```

A REDEFINES clause must immediately follow the data-name. It cannot be used with 01-level entries in the File Section.

The level numbers of both data-names must be identical, and no entry with a level number numerically lower than this level number can intervene between the data description entries. Thus, the following example illustrates an *illegal* description:

```
        05  A PIC 999.
    03  B PIC AAA.
        05  C REDEFINES A.
```

The description of the data-name being redefined cannot contain a REDEFINES or OCCURS clause; however, it can be subordinate to an entry that contains either clause.

The data types of the original item and the redefined area can differ. However, for items not at the record level (that is, items with a level number greater than 01), the redefinition must specify the same number of character positions as the original definition. For items at the record level, there is no such constraint.

You can redefine the same memory area more than once. Entries defining new data must immediately follow entries defining the area being redefined. Multiple redefinitions of the same positions must all use the data-name of the entry that originally defined the area. The following structure is legal:

```
05  A                  PIC 9999.
05  B REDEFINES A      PIC 9V999.
05  C REDEFINES A      PIC 99V99.
```

The following structure is *illegal:*

```
05  A                  PIC 9999.
05  B REDEFINES A      PIC 9V999.
05  C REDEFINES B      PIC 99V99.
```

In an FD entry, multiple 01-level entries implicitly redefine the same record area, and a REDEFINES clause is not permitted. For example:

```
FD  CUSTOMER-ACCT-FILE ...
01  PARTS-CUST-REC.
    03  CUST-NAME          PIC X(25).
    03  CUST-ADDRESS       PIC X(30).
...
01  ASSEMBLY-CUST-REC.
    03  CUST-ADDRESS       PIC X(20).
    03  CUST-NAME          PIC X(35).
```

PICTURE Clause

The PICTURE clause describes the general characteristics and editing requirements of an elementary data-item. You can specify a PICTURE clause only at the elementary item level. You must specify the clause for every elementary data-item, except one declared as USAGE IS INDEX.

The format of the PICTURE clause is:

{ PICTURE | PIC } IS *picture-string*

Picture-string consists of a maximum of 30 characters from the COBOL character set. The string specifies:

- The type of data in the data-item (alphabetic, alphanumeric, numeric, alphanumeric edited, or numeric edited)
- The size of the data-item
- The location of the decimal point in a numeric data-item
- Whether a numeric data-item contains an arithmetic sign
- Any editing to be performed on the data-item.

Symbols Used in the PICTURE Clause

The function of each PICTURE clause symbol is given in the following list:

A — A position containing only an uppercase letter or a space.

B — A position where a space will be inserted.

P — An assumed decimal scaling position. P can appear only in a sequence in the rightmost or leftmost portion of the format. The P is not counted in the length of the data-item. It is counted in determining the maximum number of digit positions in numeric or numeric edited items, or items that appear as arithmetic operands.

S — An operational sign. S can appear once in the character string and must be the leftmost character in the PICTURE string. The S is not counted in the size of an elementary item unless you specify the SEPARATE CHARACTER option of the SIGN clause.

V — The location of an assumed decimal point. V can appear once in a character string. When the assumed decimal point is to the right of the rightmost symbol in the string, the V is redundant. The V is not counted in the length of the data-item.

X — A position containing any character from the 7- or 8-bit ASCII character set.

Z — A leading numeric position that is replaced by a space when the contents of that position is zero.

9 — A position containing a digit from 0 through 9.

0 — A position where a zero will be inserted.

/ — A position where a slash will be inserted.

, — A position where a comma will be inserted. A comma cannot be the last character in the PICTURE character string.

. — In an input data-item, represents a decimal point used for alignment. In an output data-item, represents a position where a period will be inserted. The period cannot be the last character in an output PICTURE string.

You can exchange the functions of the period and the comma in a PICTURE character string by specifying the DECIMAL-POINT IS COMMA clause in the SPECIAL-NAMES paragraph.

* — A leading numeric position where an asterisk will be placed when the content of that position is zero.

$ — (or other currency symbol) A position where a $ or other one-character currency symbol will be placed.

+, -, CR, DB — These editing sign-control symbols are mutually exclusive in a character string. You can use the + and - symbols on the left or right side of the value. You can use the CR and DB symbols only on the right side.

USAGE Clause

The USAGE clause specifies the way in which a data-item is stored in memory or on disk. The USAGE clause can be written at any level except the 88 level. When written at the group level, the USAGE clause applies to all items in the group. If you specify the USAGE clause for any of the items in the group, it must not conflict with the USAGE specified at the group level.

The format of the USAGE clause is:

```
              { DISPLAY            }
              { BINARY             }
              { COMP               }
[ USAGE IS ]  { COMPUTATIONAL      }
              { PACKED-DECIMAL     }
              { INDEX              }
              { COMPUTATIONAL-3    }
              { COMP-3             }
```

DISPLAY indicates that data is stored in ASCII format. If you do not specify a USAGE clause for an elementary item, USAGE IS DISPLAY is the default.

The PICTURE of a BINARY, COMPUTATIONAL, or PACKED-DECIMAL data-item can contain one or more 9s, the operational sign symbol S, the implied decimal point symbol V, and Ps. You can't have a PICTURE clause that permits editing. COMP is an abbreviation for COMPUTATIONAL.

When you describe a group as COMPUTATIONAL, BINARY, or PACKED-DECIMAL, the elementary items in the group are COMPUTATIONAL, BINARY, or PACKED-DECIMAL. The group item itself is treated as DISPLAY and cannot be used in arithmetic operations.

NOTE: You can use BINARY, PACKED-DECIMAL, and COMPUTATIONAL-3 only if you are using code revision 9 or greater.

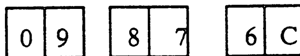
BINARY specifies that the data is stored in binary format. BINARY always gives a size error when the decimal representation of the value exceeds the number of digits in the PICTURE clause. BINARY is equivalent to COMPUTATIONAL when you compile your program with the compiler switch that causes COMPUTATIONAL items to behave as described in the ANSI '74 standard.

COMPUTATIONAL items are represented internally in twos-complement binary notation. Storage for COMPUTATIONAL items is more efficient than for DISPLAY items. Interactive COBOL lets you access the largest number that fits into the allocated number of bytes specified by the COMP field. For example, a PIC 9(4) COMP field can hold a number up to 65535.

However, if you use the compiler switch that causes COMPUTATIONAL items to behave as described in the ANSI '74 standard, you can access a number up to 9999 only. The number is truncated if you move it to a DISPLAY field that is less than PIC 9(4). To access a number greater than 9999, you must define a field of 9(5) COMP or greater. Also, Interactive COBOL detects ON SIZE ERROR conditions when the decimal value of the result is greater than the number of digits specified in

the PICTURE clause for data-items of COMP usage. Without this switch, COMP looks at the size of the item, not the PICTURE clause, and gives an ON SIZE ERROR if the number of bytes in the binary number are greater than the number allocated by the COMPUTATIONAL item. Table 7-3 lists the bytes required to store COMPUTATIONAL items.

PACKED-DECIMAL, COMPUTATIONAL-3, and COMP-3 items use binary-coded decimal form with the numeric items being represented in radix 10. Each byte of the internal representation contains two digits. The sign is always present and is represented as a separately trailing sign. The sign digit for a positive value is hexadecimal C; for a negative value, hexadecimal D; and for an unsigned value, hexadecimal F. If the picture clause for the item contains an even number of digits, the internal representation will have an extra leftmost digit that has a value of 0. If you don't want this extra digit to be considered for size error checking, then you should use the compiler switch that produces ANSI-compatible code when you compile your code.The PACKED-DECIMAL representation of +9876 is:



USAGE IS INDEX indicates that the item contains a binary value used to reference a particular element of a table. Items that are INDEX are represented internally as two-byte unsigned items. The SYNCHRONIZED, JUSTIFIED, PICTURE, VALUE IS, and BLANK WHEN ZERO clauses cannot be used with an item described as USAGE IS INDEX.

Table 7-3 COMPUTATIONAL Item Storage

No. of Decimal Digits		
Unsigned	**Signed**	**Bytes Required**
1-2	1-2	1
3-4	3-4	2
5-7	5-6	3
8-9	7-9	4
10-12	10-11	5
13-14	12-14	6
15-16	15-16	7
17-18	17-18	8

SIGN Clause

Use the SIGN clause to explicitly state the manner of sign representation in numeric data-items. The format of the SIGN clause is:

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

Specify the SIGN clause only for an elementary numeric data-item with a PICTURE containing an S, or for a group item containing at least one such numeric data-item. A data-item can have only one SIGN clause and must be described, implicitly or explicitly, as USAGE IS DISPLAY.

The terms used in this clause and their meanings are:

Sign Clause	Sign Type
SIGN LEADING	Decimal with leading sign overpunch
SIGN TRAILING	Decimal with trailing sign overpunch (default)
SIGN LEADING SEPARATE	Decimal with leading separate sign
SIGN TRAILING SEPARATE	Decimal with trailing separate sign

A numeric data-item can be from 1 to 18 digits long. If you include the SEPARATE CHARACTER phrase, the S in the PICTURE string is counted in the length of the data-item. If you omit the SEPARATE CHARACTER phrase, the S is not counted.

In decimal data-items with trailing sign overpunch, the sign is in the rightmost digit. With leading sign overpunch, the sign is in the leftmost digit (see Table 7-4).

Table 7-4 Sign Overpunch Characters

Digit	Positive	Negative
0	{	}
1	A	J
2	B	K
3	C	L
4	D	M
5	E	N
6	F	O
7	G	P
8	H	Q
9	I	R

The following program illustrates how the SIGN clause affects data-item display:

```
PROGRAM-ID. TESTPROG.
...
WORKING-STORAGE SECTION.
77  DATA-1  PIC S999 SIGN IS LEADING SEPARATE
                     VALUE +470.
77  DATA-2  PIC S999 SIGN IS LEADING
                     VALUE +470.
77  DATA-3  PIC S999 SIGN IS TRAILING SEPARATE
                     VALUE +470.
77  DATA-4  PIC S999 SIGN IS TRAILING
                     VALUE +470.

PROCEDURE DIVISION.
DISPLAY-PARA.
DISPLAY DATA-1.
DISPLAY DATA-2.
```

```
DISPLAY DATA-3.
DISPLAY DATA-4.
STOP RUN.
```

When you execute TESTPROG, the following appears on the screen:

```
+470
D70
470+
47{
```

OCCURS Clause

The OCCURS clause defines tables and other sets of repeated data-items. Within a table description, the OCCURS clause specifies the number of times the named item will be repeated. All data description clauses associated with an item containing an OCCURS clause apply to each occurrence of the item.

The format of an OCCURS clause is:

```
OCCURS integer TIMES
[; INDEXED BY { index-name }...]

[ { ASCENDING  } KEY IS { data-name }... ] ...
  { DESCENDING }
```

Do not use the VALUE clause in a data description entry that contains, or is subordinate to, an OCCURS clause. Do not specify the OCCURS clause at the 01 or 77 level.

The data-name that is the subject of an OCCURS clause must either be subscripted or indexed when referred to in a Procedure Division statement. In addition, if the data-name of the entry is the name of a group item, all data-names belonging to the group must be subscripted or indexed when they are referred to, except when they are objects of a REDEFINES clause. When the data-name is the object of a REDEFINES clause, it represents every occurrence of that name and therefore cannot be subscripted or indexed.

The INDEXED BY phrase is required when the data-name in an OCCURS clause or its subordinate items will be referenced by indexing. Each *index-name* must be a unique name within the program. Index names are not explicitly defined elsewhere. Specifying a name in the INDEXED BY phrase implicitly defines the name as a two-byte unsigned binary item.

The ASCENDING/DESCENDING KEY IS phrase specifies the order in which the system searches the keys.

Use nested OCCURS clauses to define multidimensional tables. An OCCURS clause is nested when an item containing an OCCURS clause is subordinate to another item containing an OCCURS clause. Three levels of nesting are permitted. Thus, a table can have one, two, or three dimensions.

SYNCHRONIZED Clause

The SYNCHRONIZED clause is ignored and has no effect on compilation or execution of the object program. SYNC is an abbreviation for SYNCHRONIZED.

The format of the SYNCHRONIZED clause is:

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

JUSTIFIED Clause

The JUSTIFIED clause specifies nonstandard positioning of data within a receiving data-item. JUST is an abbreviation for JUSTIFIED.

The format of the JUSTIFIED clause is:

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

You can specify this clause only at the elementary item level. It cannot be specified for a numeric item or for an item whose PICTURE clause specifies editing.

When you describe a receiving item with the JUSTIFIED clause and the sending item is larger, the leftmost characters are truncated. When the receiving item is larger, the data aligns at the right, and the leftmost positions fill with spaces (see Figure 7-1).

BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause substitutes spaces for a numeric or numeric edited item when its value is zero.

The format of BLANK WHEN ZERO is:

$\underline{\text{BLANK}}$ WHEN $\underline{\text{ZERO}}$

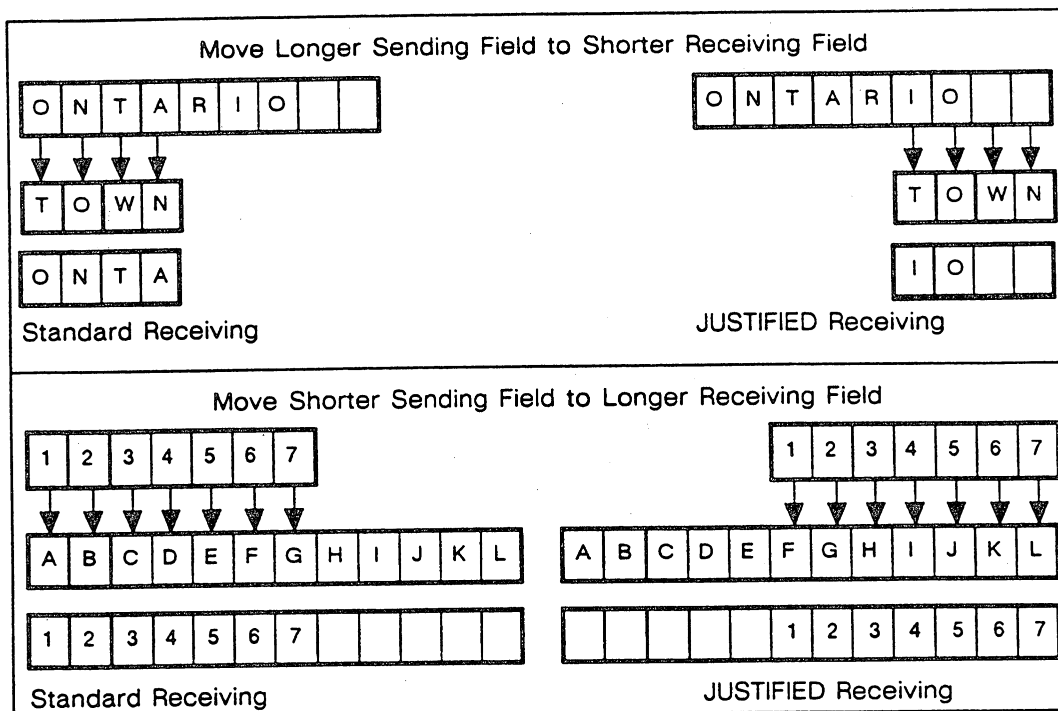


Figure 7–1 Moving Data to Standard and Justified Fields

A numeric item is considered numeric edited when its description contains the BLANK WHEN ZERO clause. This clause can only be used with data items whose usage is DISPLAY.

VALUE Clause

The VALUE clause defines the values of constants and the initial values of Working-Storage and Screen Section data-items. It cannot be used in the File or Linkage sections. The format is:

VALUE IS *literal*

If the VALUE clause occurs in Working-Storage, you can substitute a figurative constant for the literal. Figurative constants cannot be used in the Screen Section.

If the VALUE clause is not associated with a data item, the initial value of that data item is undefined. Since the initial value of any data-item is unpredictable, you should first initialize the data-item with a VALUE clause or with a Procedure Division statement. You cannot define the initial value of an index item with the VALUE clause. You must set the value with the SET statement.

You cannot use the VALUE clause in a data description entry that contains, or is subordinate to, an entry containing a REDEFINES or an OCCURS clause.

The VALUE clause must not conflict with other clauses in the data description of the item. If the category of the item is numeric, the VALUE literal must be numeric. The value of a numeric literal must be within the range of values indicated in the associated PICTURE clause, and it must not have a value that requires truncating nonzero digits. The literal can be signed only if the PICTURE clause contains a sign.

If a data-item is alphabetic, alphanumeric, alphanumeric edited, or numeric edited, the VALUE literal must be nonnumeric, with one exception: an item with a PICTURE clause containing only Zs and 9s can have a numeric value. For example, you can describe an item as PIC ZZZ9 VALUE 1234. The value of the literal must not exceed the size indicated by the PICTURE clause.

Editing characters in the PICTURE clause are included in determining the length of the data-item but do not affect initialization. When you specify a VALUE clause with a JUSTIFIED or BLANK WHEN ZERO clause, the VALUE literal is neither justified nor blank when zero. These clauses do not take effect until you move a value to the data-item.

If you use the VALUE clause 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. You cannot use the VALUE clause at subordinate levels within such a group.

End of Chapter

Chapter 8
The Procedure Division

The Procedure Division is required in every COBOL source program. It contains the operational statements that govern program input and output, data movement, logic or sequence control, and arithmetic.

The Procedure Division can be in declarative or nondeclarative format. In declarative format, you specify procedures used in the event of I/O errors. For more information on declaratives, see Chapter 1 and the USE statement later in this chapter.

Declarative Format

```
PROCEDURE DIVISION [ USING {id}... ].

[ DECLARATIVES.

{ section-name SECTION [segment-number].     }
{      USE sentence                          }  ...
{ { paragraph-name.  [sentence]...}...       }

END DECLARATIVES. ]

{ section-name SECTION [segment-number].     }
{ paragraph-name.  [sentence]...             }  ...
```

Nondeclarative Format

```
PROCEDURE DIVISION [ USING {id}... ].

{ [section-name SECTION [segment-number] ] ....  }
{ paragraph-name.  [sentence]...                 }  ...
```

Segment-number is used for documentation only.

Terminating Procedure Division Statements

You can terminate certain Procedure Division statements explicitly or implicitly. Explicit scope delimiters are any COBOL verbs that accept the END- prefix. These verbs are:

END-ACCEPT	END-ADD	END-CALL	END-COMPUTE
END-DELETE	END-DISPLAY	END-DIVIDE	END-IF
END-MULTIPLY	END-PERFORM	END-READ	END-REWRITE
END-SEARCH	END-START	END-STRING	END-SUBTRACT
END-UNDELETE	END-UNSTRING	END-WRITE	

NOTE: Interactive COBOL revisions prior to 1.60 do not support END- verbs. Interactive COBOL revision 1.60 supports END- verbs only if you include the appropriate compiler switch when you compile your program. See your user's manual for information on compiler switches. With Interactive COBOL revisions 1.70 and greater, you don't need to include a compiler switch to enable END- verbs.

Explicit scope delimiters change conditional statements into imperative statements. Thus, you can nest statements that contain phrases such as ON SIZE ERROR. Interactive COBOL pairs each END-*verb* with the preceding *verb* that is at the same level of nesting. For example, if you compile your program with the appropriate compiler switch, then the following syntax is legal in Interactive COBOL revision 1.60:

```
ADD A TO B
   ON SIZE ERROR
       MULTIPLY D BY C
          ON SIZE ERROR
              MOVE 20 TO STAT
       END-MULTIPLY.
```

To terminate a statement implicitly, you can:

- End a sentence with period.
- Include a phrase, such as ELSE, that indicates the end of the statement contained within the preceding phrase.

Procedure Division Statements

This chapter describes the Interactive COBOL procedure statements, arranged alphabetically by the verb that begins the statement. The purpose and format of each statement is presented, along with a discussion of execution features.

ACCEPT

Transfers data from the screen to the File, Working-Storage, or Linkage sections, or transfers system data into a specified data-item.

Screen Format

```
                       [    { LINE id-lit [ { COLUMN } id-lit ] } ]
ACCEPT screen-name     [ AT {             [ { COL    }        ] } ]
                       [    { { COLUMN } id-lit                 } ]
                       [    { { COL    }                        } ]

    [TIME-OUT AFTER id-lit] [; ON ESCAPE imper-stmt] [ END-ACCEPT ]
```

Identifier Format

```
ACCEPT id [TIME-OUT AFTER id-lit] [; ON ESCAPE imper-stmt]
    [ END-ACCEPT ]
```

System Format

```
                  { DATE             }
                  { DAY              }
                  { TIME             }
ACCEPT id FROM    { LINE NUMBER      }
                  { USER NAME        }
                  { ESCAPE KEY       }
                  { EXCEPTION STATUS }
                  { ENVIRONMENT      }
```

Examples

```
ACCEPT INFO-SCREEN AT LINE LINE-NUM COL COL-NUM
    ON ESCAPE GO TO FUNCTION-ROUTINE.
ACCEPT TEMP-AMT.
ACCEPT TEMP-AMT TIME-OUT AFTER 30.
ACCEPT DATE-TODAY FROM DATE.
ACCEPT FUNCTION-CODE FROM ESCAPE KEY.
ACCEPT EXCEP-CODE FROM EXCEPTION STATUS.
```

Rules for Screen ACCEPT

Screen names cannot be subscripted.

The ACCEPT *screen-name* statement transfers data entered on the screen to the data-items associated with the screen name. The program should execute a DISPLAY before an ACCEPT to display any associated prompts. ACCEPT verbs with variable origins (that is, those with the AT LINE *n* COL *m* clauses) are zero based. For example, if *screen-name* begins at line 1 column 1, ACCEPT *screen-name* AT LINE 1 COLUMN 1 will accept the screen at line 2 column 2. Thus, the simple ACCEPT *screen-name* is equivalent to the variable origin ACCEPT *screen-name* AT LINE 0 COLUMN 0.

The LINE and COLUMN clauses enable you to accept a screen name with a specified origin. The LINE and COLUMN identifiers or literals must be elementary integers. If you execute an ACCEPT with an optional LINE or COLUMN value that is larger than your screen, the screen will wrap; that is, if a display is 24 lines by 80 columns, LINE and COLUMN values of 25 and 81 are equivalent to 1 and 1.

If you display a screen with the LINE and COLUMN options, you must accept it using the same coordinates.

The words COLUMN and COL are interchangeable.

The TIME-OUT AFTER *id-lit* phrase enables a local time-out. This phrase specifies the amount of time in seconds that Interactive COBOL will wait between keystrokes as you enter the field; it does not specify the amount of time that you have to enter the entire field. The value for *id-lit* must be from 1 through 65535. If the value is 0 or a negative number, an error occurs and the time-out is disabled. Any value greater than 65535 causes the time-out to default to 65535. When a time-out occurs, Interactive COBOL places a value of 99 in the ESCAPE KEY code and the ACCEPT terminates as if you had pressed the Escape key. The TIME-OUT phrase is available only with code revision 8 or greater.

NOTE: The time-out value you specify for *id-lit* in the ACCEPT statement overrides any time-out value specified globally for your runtime system. See your user's guide for information on setting global time-outs. Your user's guide also contains information on enabling a shutdown or abort option that is associated with global time-outs. (You can enable time-outs only if you are using Interactive COBOL revision 1.60 or greater.)

The section "Data Movement with DISPLAY and ACCEPT" in Chapter 3 lists details about the execution of an ACCEPT *screen-name* statement.

Moving data from the Screen data-item to the File, Working-Storage, or Linkage data-item follows standard COBOL rules, with one exception: a move from a numeric edited to a numeric data-item is allowed. In this case, all characters of the numeric edited data-item are ignored except digits, a sign, and a decimal point. The decimal point is used for alignment only.

If you press the Escape key at any time during an ACCEPT statement, the data in the current field is not validated or transferred to screen storage. However, the contents of screen storage is still moved to the associated data-items in the File, Working-Storage, or Linkage sections. The ACCEPT terminates and the ON ESCAPE *imper-stmt*, if present, executes.

The maximum size of a field within a screen is 128 characters.

You can use the END–ACCEPT phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the ACCEPT statement. (With revisions prior to 1.70, you need to enable END–ACCEPT by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

Rules for Identifier ACCEPT

The ACCEPT *identifier* statement transfers data entered on the screen to the specified data-item, according to the rules of the MOVE verb. ACCEPT is usually preceded by a DISPLAY statement that prompts the user for the required information. For example:

```
DISPLAY "Enter the purchase price:".
ACCEPT PURCHASE-PRICE.
```

You must input data valid for the identifier. If the data is invalid because of sign, length, decimal point, or characters that do not agree with the PICTURE clause, the system displays an error message and you can correct the input.

The TIME-OUT clause works the same way on data-item ACCEPT statements as it does on screen ACCEPT statements.

Rules for System ACCEPT

DATE, DAY, TIME, LINE NUMBER, USER NAME, ESCAPE KEY, and EXCEPTION STATUS are system defined data-items treated as elementary items. ENVIRONMENT is a system defined data-item treated as a group item. These items are not explicitly defined in COBOL programs. The identifier used with a system ACCEPT must have a PICTURE clause that agrees with the value of the system defined item.

DATE is composed of year, month, and day. It can be described as PIC 9(6) or PIC X(6). For example, July 1, 1990, is expressed as 900701.

DAY is composed of year and day of year in Julian format. Its PICTURE is 9(5). For example, July 1, 1990, is expressed as 90183.

TIME is composed of hours, minutes, seconds, and hundredths of a second, and is based on elapsed time after midnight on a 24-hour clock. Its PICTURE is 9(8). Thus 2:41 P.M. is expressed as 14410000. The seventh and eighth digits, representing hundredths of a second, are always 00.

LINE NUMBER contains the four-digit number of the terminal line on which the program is running. Its PICTURE is 9(4).

USER NAME returns the user name associated with the current process, or spaces on systems where this is not meaningful. It's PICTURE is X(15). See your user's guide for more information.

The ESCAPE KEY item contains a two-digit code generated by a termination key (see "Termination Codes" later in this section). Its PICTURE is 9(2).

The EXCEPTION STATUS item contains a three-character code that identifies the type of exception condition that has occurred during the execution of a CALL or

CALL PROGRAM statement. Its PICTURE is 9(3). A table of Exception Status codes appears in Appendix A of each user's guide.

You should examine EXCEPTION STATUS immediately after the CALL or CALL PROGRAM statement. File I/O operations will alter the Exception Status value, making it undefined.

The ENVIRONMENT option transfers operating system environment data to a specified data structure. The amount of data transferred depends on the environment and the revision of the runtime system. The data structure includes the following items:

SYSTEM-CODE
: A two-digit field identifying the operating system where the Interactive COBOL programming is executing. The following is a list of system codes:
  01 AOS/VS
  02 multi-user MS-DOS
  03 single-user MS-DOS
  04 AOS/VS II
  05 DG/UX
  06 Interactive UNIX Systems

REVISION-CODE
: A two-digit field identifying the revision level of the environment information. This code changes whenever information is added to the ENVIRONMENT data structure. The data structure is upwardly compatible. Code written using one version of the data structure will operate successfully with future revisions. Additional values for existing items can be defined without changing the REVISION-CODE.

PROGRAM–NAME
: A 28-character alphanumeric field containing the simple program name (not a fully qualified name) of the program that is executing.

PID
: A five-digit numeric field containing the process–id. On systems without process–id numbers, this value is equivalent to the console number.

CONSOLE–TYPE
: A one-character alphanumeric field identifying the console environment. The values returned are defined differently for each operating system.

The ENVIRONMENT option is available only with code revision 7 or greater.

Terminating Screen Input

Depending on your operating system and your keyboard, you can terminate input by pressing the following keys: New Line, CR, Null, Form Feed, down-arrow, Tab, Esc, enter and enter-minus keys, and function keys. (See your user's guide for more information.)

To use the Escape key or function keys for terminating input, you must code a standard ACCEPT statement before coding an ACCEPT...FROM ESCAPE KEY. For example:

```
WORKING-STORAGE SECTION.
    ...
    77  DECISION-KEY PIC 99.
    ...
PROCEDURE DIVISION.
    ...
    ACCEPT DECISION-KEY-ENTRY.
    ACCEPT DECISION-KEY FROM ESCAPE KEY .
    IF DECISION-KEY = 02 PERFORM F1-ROUTINE.
```

Termination Codes

When you press a terminator key, the system moves a two-digit value to the ESCAPE KEY identifier. The item can then be interrogated by the program to determine which key was struck. The codes generated are listed in Table 8–1.

Function keys, CR, New Line, Ctrl, and Cmd function differently depending on the keyboard and the operating system. For information on the number, location, and use of the various keys, see the appropriate terminal operating manual.

Your keyboard may not contain all the keys listed in Table 8–1. For keys that are implemented, all environments return the same ESCAPE KEY codes, except for F9, F10, and F11 which return the values 34, 35, and 36 under the UNIX, AOS/VS, and MS-DOS systems. On some terminals F9, F10, and F11 are equivalent to the keys labeled (–), ENTER, and ENTER –.

NOTE: If an ACCEPT statement times out, the runtime system returns an ESCAPE KEY code of 99.

Table 8-1 Escape Key Codes

	Key Alone	Key+SHIFT	Key+Ctrl	Key+Shift+Ctrl
CR	00	00	00	00
New Line	00	00	00	00
Down-arrow	00	00	00	00
Up-arrow*	76	70	76	70
Right-arrow	n/a	71	n/a	71
Left-arrow	n/a	72	n/a	72
Cmd+Print	73	74	73	74
Tab	00	00	00	00
Esc	01	01	01	01
F1	02	10	18	26
F2	03	11	19	27
F3	04	12	20	28
F4	05	13	21	29
F5	06	14	22	30
F6	07	15	23	31
F7	08	16	24	32
F8	09	17	25	33
F9**	34	41	48	55
F10**	35	42	49	56
F11**	36	43	50	57
F12	37	44	51	58
F13	38	45	52	59
F14	39	46	53	60
F15	40	47	54	61
C1	62	66	62	66
C2	63	67	63	67
C3	64	68	64	68
C4	65	69	65	69
HOME***	n/a	75	n/a	75

* Up-arrow returns a 76 only if you use the appropriate behavior available for each runtime system. See your user's guide for more information.

** This ESCAPE KEY value is returned under UNIX and MS-DOS, and under AOS/VS when you use the global F switch.

*** This key is supported only under UNIX, AOS/VS, and MS-DOS.

ADD

Sums two or more numeric data-items and stores the result.

Cumulative ADD Format

ADD { *id-lit*}... TO { *id* [ROUNDED]} ... [; ON SIZE ERROR *imper-stmt*]
    [END-ADD]

Totaling ADD Format

ADD {*id-lit*} ... TO { *id-lit*}... GIVING { *id* [ROUNDED]} ...
    [; ON SIZE ERROR *imper-stmt*] [END-ADD]

Examples

```
ADD REG-HRS, OT-HRS TO TOT-HRS; ON SIZE ERROR GO TO HRS-ERR-RTN.
ADD NEW-STOCK, OLD-STOCK GIVING TOT-STOCK.
```

Rules

The identifiers in the cumulative or totaling ADD must refer to elementary numeric items. However, each identifier following the word GIVING can refer to an elementary numeric or numeric edited item. (If you use code revision 8, you can have multiple destinations in arithmetic statements.)

Literals must be numeric.

A SIZE ERROR will occur during the execution of an ADD statement if any intermediate result exceeds 18 digits. A size error also occurs if the sum overflows the receiving item after decimal point alignment.

With cumulative ADD, the values of the data-items before the word TO are added together. This sum is then added to the current value of the identifier following the word TO, and the result is stored as the new value of that identifier. This process is repeated for each identifier that follows the word TO. For example, in the sentence:

```
ADD A, B TO C, D.
```

the sum of A+B is added to the value in C, and this result is stored in C. Then the sum of A+B is added to the value in D and this result is stored in D.

With totaling ADD, the values of the data-items preceding the word GIVING are added together, and the sum is then stored as the new value of the identifier following the word GIVING. This process is repeated for each identifier that follows the word GIVING. For example, in the sentence:

```
ADD A, B GIVING C, D.
```

the value in A is added to the value in B, and the result is stored in C and in D.

You can use the END-ADD phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the ADD statement. (With revisions prior to 1.70, you need to enable END-ADD by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

ADD CORRESPONDING

Adds identically named items in two groups.

Format

ADD { CORRESPONDING | CORR } *id* TO *id* [ROUNDED] [; ON SIZE ERROR *imper-stmt*]

[END-ADD]

Examples

```
ADD CORRESPONDING LIST1 TO LIST2.
ADD CORR NEW-INVEN TO CURR-INVEN; ON SIZE ERROR GO TO ROUTINE3.
```

Rules

Both identifiers must refer to group items. The elements to be added must all be elementary items and none may be designated as FILLER, REDEFINES, OCCURS, or USAGE IS INDEX. Level 77 or level 88 items cannot be used. No subgroups can be added together.

The groups need not contain exactly the same list of identifiers. ADD CORRESPONDING compares the elements that make up each group. Those that have the same name are added together, and the result is stored in the second operand. All others are ignored.

If a SIZE ERROR occurs on any of the additions, all remaining additions are performed before the imperative statement is executed.

The following examples illustrate the use of ADD CORRESPONDING:

```
01  LIST1.                          01  LIST2.
    02  APPLES     PIC 99.              02  LEMONS     PIC 99.
    02  ORANGES    PIC 99.              02  APPLES     PIC 99.
    02  PEARS      PIC 99.              02  ORANGES    PIC 99.
```

The initial values are:

```
LIST1                               LIST2
    APPLES     5                        LEMONS     2
    ORANGES   10                        APPLES     6
    PEARS      8                        ORANGES    3
```

After execution of ADD CORRESPONDING LIST1 TO LIST2 the values are:

```
LIST1                               LIST2
    APPLES     5                        LEMONS     2
    ORANGES   10                        APPLES    11
    PEARS      8                        ORANGES   13
```

The CORRESPONDING phrase does not require that the level numbers be identical. However, the record structures must be the same. For example, consider these data descriptions:

```
01  LIST1.                          01  LIST2.
    02  APPLES      PIC 99.             02  SUBLIST2.
    02  ORANGES     PIC 99.                 03  LEMONS      PIC 99.
    02  PEARS       PIC 99.                 03  APPLES      PIC 99.
                                            03  ORANGES     PIC 99.
```

The statement ADD CORRESPONDING LIST1 TO SUBLIST2 is legal. The statement ADD CORRESPONDING LIST1 TO LIST2 generates a warning message stating that no items in the two groups correspond.

You can use the END-ADD phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the ADD CORRESPONDING statement. (With revisions prior to 1.70, you need to enable END-ADD by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

CALL

Transfers control to another program.

Format

CALL *id-lit* [USING { *data-name*}...] [; ON { EXCEPTION | OVERFLOW } *imper-stmt*]

[END-CALL]

Examples

Calling program:

```
WORKING-STORAGE SECTION.
01  DATA-1      PIC XX.
    77  DATA-2  PIC XXX.
PROCEDURE DIVISION.
    MOVE "ADSYS1" TO PROG-NAME.
    CALL PROG-NAME USING DATA-1, DATA-2
        ON EXCEPTION PERFORM OOPS.
    DISPLAY DATA-1, DATA-2.
```

Called program:

```
PROGRAM-ID.  ADSYS1.
...
DATA DIVISION.
...
LINKAGE SECTION.
01  DATA-ONE     PIC 99.
01  DATA-TWO     PIC 999.
...
PROCEDURE DIVISION USING DATA-ONE, DATA-TWO.
...
EXIT-PARA.
    EXIT PROGRAM.
```

Rules

The ON OVERFLOW and ON EXCEPTION phrases are functionally identical and mutually exclusive. If the call is unsuccessful, the ON OVERFLOW or ON EXCEPTION phrase transfers control to *imper-stmt*. If you omit the phrase, control passes to the statement following the CALL statement.

Id-lit can be an alphanumeric or numeric identifier, or a numeric or nonnumeric literal. Its value must be a valid external filename.

Upon encountering a CALL statement, the runtime system first determines whether the called program is executable. If it is, the runtime system transfers control to the new program, but it does not close any open files. A run unit can contain a maximum of 16 programs (one main program and 15 subprograms). The CANCEL statement removes a program from the run unit.

The USING phrase passes data to called programs. You can pass up to 32 parameters in the CALL statement. You can specify CALL...USING only if there is a Procedure Division Using header in the called program. Otherwise, the CALL fails and the system returns an exception status code. The CALL...USING and the Procedure Division Using header must contain the same number of identifiers, since a one-to-one correspondence is set up between them. The corresponding identifiers must be the same size, although their PICTURE clauses do not need to be identical. You must define items in the USING list at the 01 or 77 levels.

The CALL statement cannot pass switches to a called Interactive COBOL program. Switches are inherited from the calling program automatically.

In the example given, a correspondence is set up between the two sets of identifiers so that DATA-1 and DATA-ONE refer to the same data-item, and DATA-2 and DATA-TWO refer to the same data item. Therefore, if the called program modifies DATA-ONE and DATA-TWO, the results are available in the calling program through DATA-1 and DATA-2.

The Linkage Section must appear in the called program and must contain the data descriptions of the identifiers named in the Procedure Division Using header.

If you call a program more than once, the data-items in the called program remain as they were when the program last exited, unless the program has been cancelled. In particular, data-items whose descriptions contained a VALUE clause will not be reinitialized with that value, but will remain unchanged from their last reference. Recursive calls are not allowed. If PROG1 calls PROG2, PROG2 cannot call PROG1. However, PROG2 can call a third program, PROG3. PROG3 cannot, in turn, call PROG1 or PROG2. If you attempt to do this, the CALL fails and the system returns an exception status code.

The EXIT PROGRAM statement returns control to the calling program. Control is passed to the next statement in the calling program (the DISPLAY statement, in the example above).

You can use the END–CALL phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the CALL statement. (With revisions prior to 1.70, you need to enable END–CALL by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

Calls to Executable Programs and Foreign Language Subroutines

Interactive COBOL allows you to call programs written in languages other than Interactive COBOL. These programs can be executable program files for your operating system environment, or subroutines linked into the Interactive COBOL runtime system. See your user's guide for complete information about calls to these types of programs.

CALL PROGRAM

Chains to another COBOL program or performs a system function.

Format

CALL PROGRAM *id-lit* [USING *data-name*...] [; ON {EXCEPTION | OVERFLOW} *imper-stmt*]

[END-CALL]

Examples

```
CALL PROGRAM "ADSYSO/A".
CALL PROGRAM "ADSYSO -a".          (UNIX systems only)
CALL PROGRAM PROG-NAME.
CALL PROGRAM "ADSYS1" USING PASS-DATA-1, PASS-DATA-2.
CALL PROGRAM "ADSYS1" ON EXCEPTION PERFORM CANT-FIND-IT.
CALL PROGRAM "#L".
```

Rules

Id-lit must be an alphanumeric identifier or nonnumeric literal. Its value must be a valid external filename. *Id-lit* can include from 1 to 26 switches. See "Switch-Status Condition" in Chapter 1 for information about setting and testing switches within an Interactive COBOL program.

The successful transfer of control to a called program is equivalent to the execution of a STOP RUN statement within the calling program followed by the startup of the called program. Thus you cannot return to the original (calling) program, except when performing system calls.

Upon encountering a CALL PROGRAM statement, the runtime system first determines whether the called program is executable. If it is, the runtime system closes all open files and passes control to the new program.

If the call is unsuccessful, control passes to the ON EXCEPTION imperative statement, if you specify one. If an ACCEPT ... FROM EXCEPTION STATUS clause follows the CALL PROGRAM statement, the system places a value in the EXCEPTION STATUS data-item. (See the table of Exception Status codes in your user's guide.) If you omit the ON EXCEPTION clause, control passes to the statement following the CALL PROGRAM statement.

You can use the END-CALL phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the CALL PROGRAM statement. (With revisions prior to 1.70, you need to enable END-CALL by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.

The USING Phrase

Data is passed to the called program with the USING phrase. You must define sending data names as 01 or 77-level entries in the File Section or Working-Storage Section of the calling program, and list them in the CALL PROGRAM USING statement. You must define corresponding receiving data-names in the Linkage Section, and list them in the same order in the Procedure Division Using header of the called program. Sending-receiving pairs need not have the same names or internal structures, nor do they need to be the same total length. However, if the receiving area is larger than the sending area, the receiving area is padded with LOW-VALUES. If the receiving area is smaller than the sending area, the data sent is truncated.

A USING data-name cannot be subscripted. The same data-name can appear more than once in a CALL PROGRAM USING statement. The same data-name cannot appear more than once in the Procedure Division Using header.

The following example illustrates the CALL PROGRAM USING statement.

```
PROGRAM-ID. MAINPROG.
...
DATA DIVISION.
...
WORKING-STORAGE SECTION.
01  EMP-NAME          PIC X(20).
01  ITEM-TABLE.
    02  A-ITEM OCCURS 12 TIMES.
        03  ALPHA     PIC X(10).
        03  BETA      PIC 99.
01  AMOUNT            PIC 9(5).
...
PROCEDURE DIVISION.
...
    CALL PROGRAM "CALLEDPROG" USING
        EMP-NAME, ITEM-TABLE, AMOUNT
    ON EXCEPTION PERFORM
        CANT-FIND-IT.
```

```
PROGRAM-ID. CALLEDPROG.
...
DATA DIVISION.
...
LINKAGE SECTION.
01  CUSTOMER      PIC X(20).
01  REVISED-TABLE.
02 A-ITEM OCCURS 8 TIMES.
    03  ALPHA     PIC X(15).
    03  BETA      PIC 999.
01  CHARGE        PIC 9(5).
...
PROCEDURE DIVISION USING
    CUSTOMER,
    REVISED-TABLE,
    CHARGE.
...
STOP RUN.
```

Calls to Programs in Other Directories

See your user's guide for information about file search rules and access permissions.

System Calls

The USING phrase is not valid when executing system calls.

The following system calls are available on all systems:

#L	Chain to the program called LOGON.
#N	Rename a file.
#W	Wait (pause) for a specified time.
#S	Shut down the runtime system.

Your program can include system calls that are not implemented on your system. If the call is not implemented and there is an ON EXCEPTION *imper-stmt*, that statement is executed. If the ON EXCEPTION clause is missing, execution continues with the next statement. In either case, the runtime system generates an exception status code. See your user's guide for information about additional system calls.

CANCEL

Returns a program executed with a CALL to its initial state.

Format

<u>CANCEL</u> { *id-lit*}...

Example

```
MOVE "PROG1" TO IDEN.
CANCEL IDEN.

CANCEL "PROG1".
```

Rules

If you specify an identifier, its value must be a program-name. An identifier cannot be subscripted. If you specify a literal, it must be nonnumeric.

The CANCEL statement closes any open files in a called program. When you execute a CANCEL statement, a subsequent CALL to that program will find it in its initial state.

If a CANCEL statement specifies a program that has not been called or one that has already been called and cancelled, no action is taken and control passes to the next statement after the CANCEL. Any program can issue the CANCEL statement. However, a program cannot cancel itself or a program that calls it directly. For example, if PROG1 calls PROG2, PROG2 cannot cancel PROG1.

The CANCEL statement removes a program from the run unit.

CLOSE

Terminates processing of files.

Format

<u>CLOSE</u> { *filename* [WITH <u>LOCK</u>]}...

Examples

```
CLOSE CUSTOMER-FILE WITH LOCK.
CLOSE CUSTOMER-FILE, PRINTING-FILE.
```

Rules

The files referenced in the CLOSE statement need not all have the same organization or access mode.

You can execute a CLOSE statement only for a file that is open (see the OPEN statement). Following successful execution of a CLOSE statement, the record area associated with the filename contains no valid data. If the CLOSE is unsuccessful, a USE procedure, if specified, is executed.

The LOCK option prevents the file from being reopened within the program that issued the LOCK.

When you close a file with the CLOSE statement, no statement except DELETE FILE can access that file until it is opened with another OPEN statement.

COMPUTE

Assigns the value of an arithmetic expression to a single data-item.

Format

COMPUTE { *id* [ROUNDED]}... = *arith-expr* [; ON SIZE ERROR *imper-stmt*]
    [END-COMPUTE]

Examples

```
COMPUTE WAGERATE = REG * OVER + BONUS.
COMPUTE PARTS-ON-HD ROUNDED = MIN-QTY - SALES-RPT;
     ON SIZE ERROR GO TO WS.
COMPUTE MAILIST = 100 ** 5 + 4.
```

Rules

Each identifier at the left of the equal sign must refer to an elementary numeric item or an elementary numeric edited item. All other identifiers and literals must be elementary numeric items. (With code revision 8, you can have multiple destinations in arithmetic statements.)

The arithmetic expression can be a single identifier or literal. Use the COMPUTE statement to assign the value of that identifier or literal to each data-item (*id*) that precedes the equal sign. Use this assignment instead of MOVE when you want to round a numeric value or check for a size error.

The section on arithmetic expressions in Chapter 1 explains the arithmetic operators and their order of evaluation.

COMPUTE is the only COBOL statement you can use for exponentiation (see the third example above). The value of an exponent must be a single literal or identifier. Exponents cannot contain a sign or a fraction.

Data descriptions of operands need not be the same. The compiler supplies any necessary conversion and decimal point alignment throughout the calculation.

The maximum size of each operand or composite of operands is 18 digits.

Interactive COBOL truncates intermediate results of a COMPUTE operation to the precision of the final result. This could cause a problem, as in the following example:

```
01  OUT-REC      PIC 9(6).99.
...
01  A            PIC S9(6)V99.
01  C            PIC 99V99 VALUE 0.50.
01  D            PIC S9(6) VALUE 13519.
```

The statement COMPUTE A ROUNDED = C / 100 * D is evaluated as:

```
0.50 / 100 * 13519
= .005 * 13519
```

However, with the truncation of the intermediate result to the precision of the final result, S9(6)V99, the calculation performed is:

```
0 * 13519
= 0
```

If truncation poses a problem, substitute ADD, SUBTRACT, MULTIPLY, and DIVIDE statements for the COMPUTE statement. The COMPUTE statement is not more efficient than using these statements separately. Also, you can get incorrect results if you use parentheses in a COMPUTE statement. This is true because Interactive COBOL truncates intermediate results to the precision of the final result.

Again, you should break the COMPUTE statement into several smaller statements. For example, instead of:

```
COMPUTE C ROUNDED = (X / Y) * (Z / 4)
```

enter the statements:

```
DIVIDE X BY Y GIVING A ROUNDED.
DIVIDE Z BY 4 GIVING B ROUNDED.
MULTIPLY A BY B GIVING C ROUNDED.
```

You can use the END-COMPUTE phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the COMPUTE statement. (With revisions prior to 1.70, you need to enable END-COMPUTE by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

COPY

Inserts a copy of another file into the source file.

Format (UNIX, AOS/VS, AOS, DG/RDOS)

COPY *lit*

Format (AOS, DG/RDOS only)

COPY [INDEXED] *lit*

Examples

```
COPY INDEXED "INFOSCREEN".     (AOS, DG/RDOS only)
COPY "DP2:SECURITY.SR".
IF NUM > TOO-LARGE
    COPY "TOO$LARGE".
    ELSE NEXT SENTENCE.
```

Rules

The COPY literal must be nonnumeric and a valid filename for your operating system. It must be preceded by a separator, and it must end with a period. Any characters after the period in the line are ignored.

COPY INDEXED is not supported under the UNIX or AOS/VS operating systems. The keyword INDEXED indicates that the COPY file organization is indexed. The Interactive COBOL compiler treats each record of an indexed COPY file as one source line.

The maximum line length accepted by the compiler is 254 bytes for AOS/VS and UNIX systems, and 132 bytes for DG/RDOS and AOS systems.

COPY files must be in the same source entry format (CRT or card) as the source program. If COPY files are not in the same format, they do not compile correctly.

The COPY statement can appear anywhere in the source program, with three exceptions:

- It cannot appear in area A.
- It cannot appear where a comment entry is expected.
- It cannot appear where a PICTURE string is expected.

A source program can contain multiple COPY statements; however, a COPY file cannot contain a COPY statement.

When the compiler encounters a COPY statement, it reads the entire file into the source program. Compilation then resumes at the next line after the COPY statement in the original source file.

Copied text is numbered beginning with line 1, and the line numbers have a "C" suffix. You can include a compiler switch to suppress the listing of copy files. See your user's guide for information about compiler switches.

DELETE

Logically removes a record from an indexed or relative disk file, or physically removes a file from the disk.

Record DELETE Format

DELETE *filename* RECORD [; INVALID KEY *imper-stmt*] [END-DELETE]

File DELETE Format

DELETE FILE { *filename* }...

Examples

```
DELETE CUSTOMER-FILE RECORD; INVALID KEY GO TO DEL-ERR-RTN.
DELETE FILE TEMP-TRANS-FILE, TEMP-INVENTORY-FILE.
```

Rules for Record DELETE

The DELETE RECORD statement can be used only for files with indexed or relative organization.

The execution of a DELETE statement affects neither the contents of the record area in memory nor the current record pointer. Execution of a DELETE statement updates the value of the FILE STATUS item.

For sequential access, a DELETE statement must be preceded by a successfully executed READ statement for the same file. However, the program can contain non-I/O statements between the READ and DELETE statements. DELETE logically removes from the file the record retrieved by the previous READ statement. The next sequential READ retrieves the record following the deleted record.

You cannot specify the INVALID KEY phrase for a DELETE statement that references a file in sequential access mode.

For random or dynamic access, an INVALID KEY condition exists if the file does not contain the record specified by the key. You must specify either the INVALID KEY phrase or a USE procedure to handle error conditions.

When you execute a record DELETE for random or dynamic access, the system logically removes from the file the record identified by the value of the RELATIVE KEY or primary RECORD KEY associated with the filename.

A logically deleted record cannot be further accessed, although the record can be restored by executing the UNDELETE statement.

The key value of HIGH-VALUES is reserved for system use. Your program should not attempt to delete a record with a key value equal to HIGH-VALUES.

You can use the END–DELETE phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the DELETE statement. (With revisions prior to 1.70, you need to enable END–DELETE by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

Rules for File DELETE

The DELETE FILE statement physically removes the specified files from the physical devices on which they reside. Files must be closed when you execute the DELETE FILE statement.

Under UNIX and AOS/VS, if two programs open the same file, and the first program deletes the file, that file becomes unavailable to each program once that program closes the file. If you are using another operating system, the runtime system will generate a file status code on the DELETE.

Refer to your user's guide for specific information on how DELETE FILE works on your system.

DISPLAY

Transfers a group or elementary item to the screen. Performs screen positioning and special screen functions.

Screen Format

```
DISPLAY { screen-name [ AT { LINE id-lit [ { COLUMN } id-lit ] } ] } ...
                               [ { COL    }        ]
                            { { COLUMN } id-lit }
                            { { COL    }        }

    [ END-DISPLAY ]
```

Id-Lit Format

DISPLAY { *id-lit* }... [WITH NO ADVANCING] [END-DISPLAY]

Examples

```
DISPLAY INFO-SCREEN AT LINE LINE-NUM COL COL-NUM.
DISPLAY DATE-TODAY, "REPORT NUMBER", RPT-NO.
DISPLAY "INCORRECT INPUT, PLEASE RE-ENTER" WITH NO ADVANCING.
```

Rules for Screen DISPLAY

Screen-names cannot be subscripted.

The DISPLAY *screen-name* statement transfers information from data-items in the program to the display screen.

DISPLAY statements with variable origins (that is, those with the AT LINE *n* COL *m* clauses) are zero based. For example, if *screen-name* begins at line 1 column 1, DISPLAY *screen-name* AT LINE 1 COLUMN 1 displays the screen at line 2 column 2. Thus, the simple DISPLAY *screen-name* is equivalent to the variable origin DISPLAY *screen-name* AT LINE 0 COLUMN 0. The LINE and COLUMN identifiers must be integers in the range 1 to 255.

If you display a line or column value that is larger than your screen, the screen will wrap. That is, if a display is 24 lines by 80 columns, line and column values of 25 and 81 are equivalent to 1 and 1. However, no fields in a screen can begin beyond column 127.

Inclusion of the word AT indicates that LINE, COLUMN, or both are specified. The words COLUMN and COL are interchangeable.

After you execute a DISPLAY *screen-name* statement, the program must execute an ACCEPT statement to permit data entry. If you display a variable origin screen, you must accept it using the same coordinates.

When you execute a screen DISPLAY, data-items following a FROM or USING clause are moved to screen storage. If data will be entered (that is, the screen data-item has a TO clause but no FROM clause), underscores are moved to screen storage. All moves follow the COBOL MOVE rules. Once the data is moved, the screen is displayed.

Literal fields are displayed exactly as they appear in the VALUE clause in the Screen Section. The appearance of other fields is determined by the PICTURE clause, which can specify editing symbols. Table 8-2 lists the differences in appearance of the displayed fields.

After you execute a DISPLAY statement, the cursor is positioned after the last displayed field.

Table 8-2 Screen Appearance after Execution of DISPLAY

Type	Clause	Screen Appearance
Input	TO (without FROM)	Underscores defining field length (from the size of the screen data-item's PICTURE).
Output	FROM	Current value of FROM literal or data-item, edited according to the screen data-item's PICTURE.
Input/Output	USING	Current value of USING data-item, edited according to the screen data-item's PICTURE. The displayed value is available for modification when the screen data-item is accepted.

You can use the END-DISPLAY phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the DISPLAY statement. (With revisions prior to 1.70, you need to enable END-DISPLAY by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

Rules for Id-lit DISPLAY

With this format, the program cannot control placement of the values on the screen. DISPLAY places the specified literals or data values on the screen in one continuous string at the current cursor position. The cursor then moves to the next line. When you specify NO ADVANCING, the cursor remains at the end of the displayed text.

Do not use the scaling character P in a PICTURE clause, since character positions marked by P are truncated in the display data-item.

When a DISPLAY statement contains more than one operand, the length of the sending item is the sum of the lengths of the operands. DISPLAY moves the operands in the sequence in which they are encountered.

You can display any figurative constant. However you cannot use the figurative constant ALL with a DISPLAY statement. When you use a figurative constant with DISPLAY, a single occurrence displays on the screen.

DIVIDE

Divides one numeric data-item into another.

Overlaying DIVIDE INTO Format

DIVIDE *id-lit* INTO { *id* [ROUNDED]}... [; ON SIZE ERROR *imper-stmt*]
[END-DIVIDE]

Nonoverlaying DIVIDE INTO Format

DIVIDE *id-lit* INTO *id-lit* GIVING { *id* [ROUNDED]}...
[; ON SIZE ERROR *imper-stmt*] [END-DIVIDE]

DIVIDE *id-lit* INTO *id-lit* GIVING *id* [ROUNDED] [REMAINDER *id*]
[; ON SIZE ERROR *imper-stmt*] [END-DIVIDE]

Nonoverlaying DIVIDE BY Format

DIVIDE *id-lit* BY *id-lit* GIVING { *id* [ROUNDED]}...
[; ON SIZE ERROR *imper-stmt*] [END-DIVIDE]

DIVIDE *id-lit* BY *id-lit* GIVING *id* [ROUNDED] [REMAINDER *id*]
[; ON SIZE ERROR *imper-stmt*] [END-DIVIDE]

Examples

```
DIVIDE QUANTITY INTO TOTAL ROUNDED ON SIZE ERROR GO TO ERR-9.
DIVIDE QUANTITY INTO TOTAL GIVING AVERAGE ROUNDED REMAINDER AVG-R
    ON SIZE ERROR GO TO ERR-6.
DIVIDE TOTAL BY QUANTITY GIVING AVERAGE ROUNDED.
```

Rules

The identifiers representing the divisor and the dividend must refer to elementary numeric items. However, an identifier associated with the GIVING or REMAINDER phrase can refer to either an elementary numeric or numeric edited item. (With code revision 8, you can have multiple destinations in arithmetic statements.)

A literal must be a numeric literal.

When you use the overlaying DIVIDE INTO, the value of *id-lit* is divided into the value of *id*. The resulting quotient replaces the value of the dividend (*id*). Interactive

COBOL repeats this process for each dividend (*id*) that you specify. For example, in the sentence:

```
DIVIDE A INTO B, C.
```

the value in B is divided by the value in A and the result is stored in B. Then the value of C is divided by the value in A and the result is stored in C.

With the nonoverlaying DIVIDE INTO, the value of the first *id-lit* is divided into the value of the second *id-lit*, and the resulting quotient is placed in each *id* following GIVING. For example, in the sentence:

```
DIVIDE A INTO B GIVING C, D.
```

the value in B is divided by the value in A, and the result is stored in C and in D. The values in A and B do not change.

With the nonoverlaying DIVIDE BY, the value of the first *id-lit* is divided by the value of the second *id-lit*, and the resulting quotient is placed in each *id* following GIVING. For example, in the sentence:

```
DIVIDE A BY B GIVING C, D.
```

the value in A is divided by the value in B, and the result is stored in C and D. Again, the values in A and B do not change.

The REMAINDER is the product of the quotient and the divisor subtracted from the dividend. If you have a REMAINDER, you can specify only one *id* to receive the quotient. If you define the quotient identifier as a numeric edited item, the quotient used to calculate the remainder is an intermediate field that contains the unedited quotient. If you specify ROUNDED, the remainder is calculated on an intermediate value that contains the truncated, rather than rounded, value of the quotient.

The accuracy of the REMAINDER item is defined by the calculation described above. Appropriate decimal alignment and truncation, but not rounding, are performed as needed for the content of the REMAINDER data-item.

The ON SIZE ERROR phrase specifies a statement for execution in the event of a SIZE ERROR. The SIZE ERROR conditions are described in Chapter 1.

When the ON SIZE ERROR phrase is used in formats specifying REMAINDER, the following rules apply:

- If the size error occurs on the quotient, the remainder is not calculated. Thus, the contents of both the quotient identifier and the remainder identifier are unchanged.
- If the size error occurs on the remainder, the contents of the remainder identifier are unchanged.

You can use the END-DIVIDE phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the DIVIDE statement. (With revisions prior to 1.70, you need to enable END-DIVIDE by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

See "Arithmetic Expressions" in Chapter 1 for an explanation of the ROUNDED phrase.

EXIT

Documents the end of a series of procedures.

Format

EXIT.

Example

```
RTN-EXIT-PARA.
    EXIT.
```

Rules

The EXIT statement must appear in a sentence by itself, and it must be the only statement in the paragraph.

The EXIT statement lets you assign a procedure-name to a given point in a program. You may then use this reference point to define the limits of a PERFORM. An EXIT statement has no other effect on the compilation or execution of the program.

EXIT PROGRAM

Marks the logical end of a called program.

Format

EXIT PROGRAM.

Example

```
PROCEDURE DIVISION.
    ...
EXIT-PARA.
    EXIT PROGRAM.
```

Rules

EXIT PROGRAM does not close files.

The EXIT PROGRAM statement must appear in a sentence by itself, and it must be the only statement in the paragraph.

The EXIT PROGRAM statement occurs at the logical end of a called program. It causes control to return to the first statement after the CALL statement of the calling program. The called program remains in the same state as when the EXIT PROGRAM was executed.

If the EXIT PROGRAM statement occurs in a program that is not called, the following rules apply:

- If the paragraph was not performed, control passes to the first sentence in the paragraph after EXIT PROGRAM.
- If the paragraph was performed, the EXIT PROGRAM statement functions the same way as EXIT.

A called program must execute an EXIT PROGRAM statement before being cancelled. Otherwise, results are undefined.

GO TO

Transfers control from one part of the Procedure Division to another.

Simple GO TO Format

GO TO *procedure-name*

Conditional GO TO Format

GO TO *procedure-name* [; *procedure-name*...] DEPENDING ON *id*

Examples

```
GO TO BALANCE-RTN.
GO TO CHECK-1, CHECK-2, CHECK-3 DEPENDING ON CHECK-VALUE.
```

Rules

Procedure-name has the format:

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

A simple GO TO statement transfers control to the specified procedure-name. When you use a simple GO TO in a sentence containing a sequence of imperative statements, it must be the last statement in the sequence.

In the conditional GO TO format, the identifier is the name of an elementary integer data-item. A GO TO...DEPENDING statement transfers control to the procedure-name whose position in the procedure-name list corresponds to the current value of the identifier. If the current value of the identifier is less than 1 or greater than the number of procedure-names in the list, no transfer occurs and control passes to the next statement. For example:

```
GO TO PARA-10, PARA-20, PARA-30  DEPENDING  ON PARA-CHECK.
    NEXT-PARA.
```

If PARA-CHECK has a value of 1, 2, or 3, control passes to procedure-name PARA-10, PARA-20, or PARA-30, respectively. If PARA-CHECK contains any other value, control passes to the next executable statement in paragraph NEXT-PARA.

There can be no more than 255 paragraph names referenced by a GO TO...DEPENDING statement.

With either format, it is *illegal* to transfer control to:

- A procedure-name in a Declaratives section from a nondeclaratives section
- A procedure-name in a nondeclaratives section from a Declaratives section
- A Declaratives section from another Declaratives section

The use of duplicate names can cause unexpected results. The Interactive COBOL compiler makes only one pass over your program. The compiler attempts to resolve name references as they occur, taking into account that paragraph names are implicitly qualified by the section name. The following examples of COBOL program fragments demonstrate how Interactive COBOL handles ambiguous name references:

```
BB SECTION.
...
EE SECTION.
...
BB.
...
CC SECTION.
...
    GO TO BB.
```

```
AA SECTION.
...
BB.
...
BB SECTION.
...
CC SECTION.
...
    GO TO BB.
```

The examples above produce a compiler error because BB is an ambiguous procedure name. To the compiler, BB is not a paragraph in the current section. Therefore, you must qualify it if there is more than one occurrence of the name.

```
AA SECTION.
...
BB SECTION.
...
CC SECTION.
...
    GO TO BB.
BB.
...
```

The example above compiles without error. At runtime, GO TO BB causes execution of the BB paragraph. GO TO BB is interpreted as GO TO BB OF CC since the BB paragraph is in CC. This occurs because all unqualified references in a GO TO and PERFORM are first tried in the current section, and therefore are implicitly qualified. GO TO *procedure-name*, or PERFORM *procedure-name*, is never ambiguous if you declare *procedure-name* in the same section as the GO TO or PERFORM.

Licensed Material - Property of Data General Corporation

IF

Evaluates a condition and takes an action depending on whether the condition is true or false.

General Format

```
IF condition; [ THEN ] { statement-1     }
                       { NEXT SENTENCE   }

    { ; ELSE statement-2          [ END-IF ] }
    { ; ELSE  NEXT SENTENCE                  }
    {   END-IF                               }
```

Condition Formats

```
                       { GREATER THAN }
                       { LESS THAN    }
                       { EQUAL TO     }
id-lit IS [ NOT ]      { >            }  id-lit
                       { <            }
                       { =            }
                       { >=           }
                       { <=           }
```

```
                       { NUMERIC    }
                       { ALPHABETIC }
id-lit IS [ NOT ]      { POSITIVE   }
                       { NEGATIVE   }
                       { ZERO       }
```

Combined Condition Format

```
condition [ { AND } condition ]
            { OR  }
```

Abbreviated Combined Condition Format

```
relation-condition  { AND } [ NOT ] [relational-operator] object ...
                    { OR  }
```

Examples

```
IF LINE-COUNT = 60 GO TO NEW-PAGE-RTN.
IF CHOICE = 1 OR CHOICE = 2 OR CHOICE = 3 NEXT SENTENCE
    ELSE GO TO CHOICE-RTN.
IF CHOICE = 1 OR 2 OR 3 NEXT SENTENCE
    ELSE GO TO CHOICE-RTN.
IF OP-AREA EQUAL CUST-AREA PERFORM EXAMINE-RECORD
    ELSE DISPLAY "Incorrect data".
```

Rules

You can terminate an IF statement with either a period or the phrase END–IF. If you use the END–IF phrase, do not include the NEXT SENTENCE phrase.

You can omit the ELSE NEXT SENTENCE phrase if it immediately precedes the terminal period of the sentence.

When the condition stated in an IF statement is true:

- Statement-1 executes. If it contains a branching statement, control is explicitly transferred in accordance with the rules of that statement. Otherwise, statement-2 is ignored and control passes to the next executable sentence.
- If the NEXT SENTENCE phrase is used in place of statement-1, statement-2 is ignored, and control passes to the next executable sentence.

When the condition stated in an IF statement is false:

- Statement-1 is ignored and statement-2 executes. If statement-2 contains a branching statement, control is explicitly transferred in accordance with the rules of that statement. Otherwise, control passes to the next executable sentence. If statement-2 is not specified, statement-1 is ignored and control passes to the next executable sentence.
- If the ELSE clause is not specified, control passes to the next executable statement.
- If the ELSE NEXT SENTENCE phrase is specified, control passes to the next executable sentence.

When the condition is in combined form, any condition except the first can be abbreviated by omitting either:

- The subject of the relation condition
- The subject and relational operator of the relation condition

You can use the END–IF phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the IF statement. (With revisions prior to 1.70, you need to enable END–IF by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

To use the THEN phrase, you must be running Interactive COBOL revision 1.70 or greater.

Nested IF Statements

You can include IF statements within IF statements to create nested IF statements. Interactive COBOL treats nested IF statements as paired IF, ELSE, and (if included) END–IF combinations proceeding from left to right. Each ELSE or END–IF applies to the immediately preceding IF statement that has not been paired with an ELSE or END–IF. You can terminate a nested IF statement with either a period or an END–IF phrase that is at the same level of nesting.

The following example shows a nested IF statement:

```
IF INVALID-KEY-SW = "Y"
    IF TRANS-CODE = "A"
        PERFORM ADD-ROUTINE
    ELSE
        MOVE "NONMATCHING MASTER RECORD" TO ERROR-MESS
        PERFORM ERROR-ROUTINE
ELSE
    IF TRANS-CODE = "A"
        MOVE "DUPLICATE MASTER RECORD" TO ERROR-MESS
        PERFORM ERROR-ROUTINE
    ELSE
        IF TRANS-CODE = "C"
            PERFORM CHANGE-ROUTINE
        ELSE
            IF TRANS-CODE = "D"
                PERFORM DELETE-RTN.
```

This next example uses an END–IF phrase.

```
IF INVALID-KEY-SW = "N"
    IF TRANS-CODE = "C"
        PERFORM CHANGE-ROUTINE
    ELSE
        PERFORM ERROR-ROUTINE
    END-IF
    PERFORM START-OVER.
```

If you omitted the END-IF phrase, the runtime system would treat the statement PERFORM START-OVER as part of the ELSE clause instead of part of the original IF statement.

INSPECT

Counts and/or replaces specific single characters in a data-item.

Tally INSPECT Format

```
                                 { { ALL     }         } [ { BEFORE }                ]
INSPECT id TALLYING id FOR       { { LEADING } id-lit  } [ { AFTER  } INITIAL id-lit ]
                                 {   CHARACTERS        }
```

Replace INSPECT Format

```
                        { { ALL     }                   } [ { BEFORE }                ]
INSPECT id REPLACING    { { LEADING } id-lit BY id-lit  } [ { AFTER  } INITIAL id-lit ]
                        { { FIRST   }                   }
                        {   CHARACTERS BY id-lit        }
```

Tally and Replace INSPECT Format

INSPECT *id* TALLYING *tally-clause* REPLACING *replacing-clause*

Rules

The INSPECT statement must specify the TALLYING option, the REPLACING option, or both. The TALLYING identifier must be an integer.

The identifier immediately following the INSPECT verb is the item that will be inspected. It can be an elementary or group item. You must describe it as USAGE IS DISPLAY.

Identifiers or literals must be one character long and cannot be signed numeric items. When you use a figurative constant, it refers to an implicit one-character data-item.

If you specify the CHARACTERS option, an implied one-character comparison is used; the comparison is always considered a match.

If you specify the ALL option, all characters in the comparison area of the specified data-item are considered for the comparison. The LEADING option matches only contiguous occurrences of the specified character string that begin in the leftmost position of the comparison area in the data-item.

The INSPECT comparison cycle normally begins at the leftmost character of the INSPECT identifier and proceeds to the rightmost character. During comparison of the

INSPECT identifier, each matched occurrence of the string is tallied and/or replaced as specified in the REPLACING clause.

The BEFORE and AFTER phrases modify the boundaries of the INSPECT comparison. Both phrases always refer to the initial (leftmost) occurrence of the *id-lit* specified in the phrase.

When you use the BEFORE phrase, comparison continues up to, but does not include, the first occurrence of the *id-lit* specified in the BEFORE phrase.

When you use the AFTER phrase, comparison begins immediately after the first occurrence of the specified *id-lit*.

Tally INSPECT

When you use the ALL phrase, the TALLYING identifier increases by one for each occurrence of *id-lit* within the INSPECT identifier.

When you use the LEADING phrase, the TALLYING identifier increases by one for each contiguous occurrence of *id-lit* within the INSPECT identifier, provided the leftmost *id-lit* is at the point where comparison begins.

When you use the CHARACTERS phrase, the TALLYING identifier increases by one for each character in the INSPECT identifier.

Replace INSPECT

The statement must specify one of the following options: ALL, LEADING, FIRST, or CHARACTERS.

- With the ALL phrase, each occurrence of the *id-lit* preceding the BY phrase is replaced by the *id-lit* following the BY phrase.
- With the LEADING phrase, each contiguous occurrence of the *id-lit* preceding the BY phrase is replaced by the *id-lit* following the BY phrase, provided the leftmost *id-lit* is at the point where comparison begins.
- With the FIRST phrase, the leftmost *id-lit* preceding the BY phrase is replaced by the *id-lit* following the BY phrase.
- With the CHARACTERS phrase, each character in the INSPECT identifier is replaced by the character specified in the REPLACING *id-lit*.

Tally and Replace INSPECT

In this format, INSPECT functions as two separate statements, INSPECT TALLYING and INSPECT REPLACING. The system executes the INSPECT TALLYING portion first.

Sample INSPECT Statements

In the following examples the *count* data-item has value 0 before the INSPECT statement is executed.

```
INSPECT word TALLYING count FOR LEADING "L" BEFORE INITIAL "A".
```

Where word = LARGE, count = 1.
Where word = ANALYST, count = 0.

```
INSPECT word TALLYING count FOR ALL "L" REPLACING LEADING
        "A" BY "E" AFTER INITIAL "L".
```

Where word = CALLAR, count = 2, word = CALLAR.
Where word = LATTER, count = 1, word = LETTER.

```
INSPECT word TALLYING count FOR CHARACTERS AFTER INITIAL "J"
        REPLACING ALL "A" BY "B".
```

Where word = ADJECTIVE, count = 6, word = BDJECTIVE.
Where word = JACK, count = 3, word = JBCK.

```
INSPECT word REPLACING ALL "X" BY "Y" AFTER INITIAL "R".
```

Where word = REXMALL, word = REYMALL.
Where word = TEXARKANA, word = TEXARKANA.

```
INSPECT word REPLACING CHARACTERS BY "B" BEFORE INITIAL "A".
```

Where word = 12XZABCD, word = BBBBABCD.

```
INSPECT word REPLACING ALL "A" BY "G" BEFORE INITIAL "X".
```

Where word = ARXAS, word = GRXAS.
Where word = HANDAX, word = HGNDGX.

MOVE

Transfers data from one data-item to one or more data-items.

Format

MOVE *id-lit* TO { *id* }...

Examples

```
MOVE BALANCE TO LINE-3.
MOVE "THE DATE IS" TO HEAD-2.
MOVE 0 TO SUB-1, SUB-2, SUB-3.
```

Rules

Id-lit represents the sending area; *id* represents the receiving area.

Any subscripting or indexing associated with the sending item is evaluated only once, immediately before the data is moved to the first of the receiving operands.

The rules for moving data apply equally to all receiving areas. Any subscripting or indexing associated with a receiving item is evaluated immediately before the data is moved to the respective item. Consider the following statement, where B = 1 and A(1) = 2:

```
MOVE A(B) TO B, C(B).
```

This statement is equivalent to:

```
MOVE 2 TO B.
MOVE 2 TO C(2).
```

A(B) is evaluated at the start only, and C(B) is evaluated immediately before moving data into that item.

Moves Between Data Types

An elementary move occurs when both the sending and receiving items are elementary items. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, or alphanumeric edited. The editing symbols used in the PICTURE clause specify the category for an item. The data categories for literals and figurative constants are:

- Numeric literals are numeric.
- Nonnumeric literals are alphanumeric.

- The figurative constant ZERO is numeric.
- The figurative constant SPACE is alphabetic.
- All other figurative constants are alphanumeric.

All numeric receiving fields are decimal point aligned, right justified, and padded with zeros. All numeric edited receiving fields are decimal point aligned and padded with the appropriate editing characters. All other receiving fields are left justified and padded with spaces.

The following restrictions apply to elementary moves between data categories:

- A numeric edited, alphanumeric edited, or alphabetic data-item, or the figurative constant SPACE, must not be moved to a numeric or numeric edited item.
- A numeric literal, the figurative constant ZERO, a numeric item, or a numeric edited item must not be moved to an alphabetic item.
- A noninteger numeric literal or a noninteger numeric item must not be moved to an alphanumeric or alphanumeric edited item.

All other elementary moves are legal. Any necessary conversion of data from one form of internal representation to another occurs during the move, along with any specified editing in the receiving item. Table 8-3 summarizes the different types of MOVE statements.

Table 8-3 Legal MOVE Combinations

	Category of Receiving Data-item		
Sending Data-Item	**Alphabetic**	**Alphanumeric Alphanumeric Edited**	**Numeric Integer Numeric Noninteger Numeric Edited**
Alphabetic	Yes	Yes	No
Alphanumeric	Yes	Yes	Yes
Alphanumeric Edited	Yes	Yes	No
Numeric Integer	No	Yes	Yes
Numeric Noninteger	No	No	Yes
Numeric Edited	No	Yes	No

You can also perform non-elementary moves. That is, you can move a group item to an elementary item or elementary items to a group item. Interactive COBOL treats a non-elementary move as if it were an alphanumeric to alphanumeric elementary move, except that Interactive COBOL does not convert the data from one form of internal representation to another. In such a move, the receiving area is filled without consideration for the individual elementary items or the group item contained within either the sending or receiving area.

MOVE CORRESPONDING

Moves items with the same name from one group item to another group item.

Format

MOVE { CORRESPONDING | CORR } *id-1* TO *id-2*

Examples

```
MOVE CORRESPONDING TEMP-LIST TO PERM-LIST.
MOVE CORR CURR-SALES TO BACKUP SALES.
```

Rules

Id-1 and *id-2* must be group items.

For two items within each group to correspond, they must have the same name. One of the corresponding items must be elementary. The other may be group or elementary. None of the items can be designated as FILLER, REDEFINES, or OCCURS. Level 77 or level 88 items cannot be used.

The MOVE CORRESPONDING verb examines each of the group items named as operands. The names of the items subordinate to these groups are compared. Items in *id-1* are moved to items in *id-2* that have the same name.

The data is combined into one alphanumeric string when the sending item is a group. When the receiving item is a group, the sending data is allocated on a position-by-position basis, with truncation and space filling.

Consult the description of the verb MOVE to see how elementary moves are handled. The following example illustrates the use of MOVE CORRESPONDING:

```
01  LIST1.                            01  LIST2.
    02  APPLES      PIC 99.               02 LEMONS      PIC 99.
    02  ORANGES     PIC 99.               02 APPLES      PIC 99.
    02  PEARS       PIC 99.               02 ORANGES     PIC 99.
```

The initial values are:

```
LIST1                                 LIST2
    APPLES      4                         LEMONS     8
    ORANGES     9                         APPLES    13
    PEARS       2                         ORANGES    5
```

After execution of MOVE CORRESPONDING LIST1 TO LIST2 the values are:

```
LIST1                           LIST2
    APPLES      4                   LEMONS    8
    ORANGES     9                   APPLES    4
    PEARS       2                   ORANGES   9
```

The CORRESPONDING phrase does not require that level numbers be identical. However, the record structures must be the same. For example:

```
01  LIST1.                      01  LIST2.
    02  APPLES      PIC 99.         02  SUBLIST2.
    02  ORANGES     PIC 99.             03  LEMONS      PIC 99.
    02  PEARS       PIC 99.             03  APPLES      PIC 99.
                                        03  ORANGES     PIC 99.
```

The statement MOVE CORRESPONDING LIST1 TO SUBLIST2 is legal. The statement MOVE CORRESPONDING LIST1 TO LIST2 generates a warning message stating that no items in the two groups correspond.

MULTIPLY

Multiplies two numeric data-items and sets the value of a data-item equal to their product.

Overlaying MULTIPLY Format

MULTIPLY *id-lit* BY { *id* [ROUNDED]} ... [; ON SIZE ERROR *imper-stmt*]
[END-MULTIPLY]

Nonoverlaying MULTIPLY Format

MULTIPLY *id-lit* BY *id-lit* GIVING { *id* [ROUNDED]} ...
[; ON SIZE ERROR *imper-stmt*] [END-MULTIPLY]

Examples

```
MULTIPLY HOURS BY RATE; ON SIZE ERROR GO TO ERR-RTN-6.
MULTIPLY 3.1416 BY DIAMETER ROUNDED;
     ON SIZE ERROR GO TO CHECK-IT.
MULTIPLY HOURS BY RATE GIVING GROSS.
MULTIPLY DIAM BY 3.1416 GIVING CIRC ROUNDED.
```

Rules

Each *id* must refer to an elementary numeric item, except that in the nonoverlaying format each *id* following the word GIVING must refer to either an elementary numeric or numeric edited item. (With code revision 8, you can use multiple destinations in arithmetic statements.)

A literal must be a numeric literal.

In the overlaying format, the value of *id-lit* is multiplied by the value of *id* and the result is stored as the new value of *id*. This process is repeated for each identifier that follows the word BY. For example, in the statement:

```
MULTIPLY A BY B, C.
```

the value in A is multiplied by the value in B, and the result is stored in B. Then A is multiplied by the value in C, and that result is stored in C.

In the nonoverlaying format, the value of the first *id-lit* is multiplied by the *id-lit* following the word BY, and the result is stored as the new value of each identifier (*id*) following the word GIVING. For example, in the statement:

```
MULTIPLY A BY B GIVING C, D.
```

the value in A is multiplied by the value in B, and the result is stored in C and D.

See "Arithmetic Expressions" in Chapter 1 for details about the ROUNDED and SIZE ERROR phrases.

You can use the END-MULTIPLY phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the MULTIPLY statement. (With revisions prior to 1.70, you need to enable END-MULTIPY by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

OPEN

Initiates processing of a file.

Format

```
                          ⎧ INPUT { filename }...    ⎫
                          ⎪ OUTPUT { filename }...   ⎪
OPEN [ EXCLUSIVE ]        ⎨ I-O { filename }...      ⎬ ...
                          ⎩ EXTEND { filename }...   ⎭
```

Examples

```
OPEN INPUT TRANS-FILE, CUSTOMER-FILE.
OPEN EXCLUSIVE I-O MASTER-FILE-1.
OPEN INPUT TRANS-FILE OUTPUT PRINT-RPT I-O MASTER-FILE.
```

Rules

The OPEN statement makes the associated record area available to the program. It does not retrieve or release data records. You must execute the READ statement to access records, and the WRITE statement to release a record. You must first open your files before executing any other Procedure Division statements on those files, with one exception: you can use the DELETE FILE statement without opening your file.

The execution of an OPEN statement updates the value of the FILE STATUS item. When the OPEN statement is unsuccessful, the program executes the USE procedure, if specified, for the file.

Each OPEN statement must specify one of the OPEN modes: INPUT, OUTPUT, I-O, or EXTEND. The file remains in that mode until the program executes a CLOSE statement for the file, the run unit terminates, or the program is cancelled. Table 8–4 lists I/O statements used with the OPEN statement. Table 8–5 lists the results of an OPEN statement on existing and nonexisting files.

You can open a file with one OPEN mode, and reopen the file with a different OPEN mode in the same program, if you close the file (without the LOCK option) before executing a subsequent OPEN statement. After successful execution of an OPEN INPUT statement, you can read the file whether the records are locked or unlocked.

When a program opens an existing indexed or relative file for INPUT or I-O, the program must describe the file exactly as it was described when it was created (that is, record size, number of keys, etc.)

When any file is opened for INPUT or I-O, the OPEN statement sets the current record pointer to the first record in the file. For an indexed file, this is the record corresponding to the lowest primary key. If no record exists in the file, the current record pointer is set so that the first READ statement executed for the file results in an AT END condition.

Table 8-4 Permissible I/O Statements with OPEN

Access Modes	Statement	Open Mode: Input	Output	I-O	Extend
Sequential	READ	S,R,I		S,R,I	
	WRITE		S,R,I		S
	REWRITE			S,R,I	
	START	R,I		R,I	
	DELETE/UNDELETE			R,I	
Random	READ	R,I		R,I	
	WRITE		R,I	R,I	
	REWRITE			R,I	
	START				
	DELETE/UNDELETE			R,I	
Dynamic	READ	R,I		R,I	
	WRITE		R,I	R,I	
	REWRITE			R,I	
	START	R,I		R,I	
	DELETE/UNDELETE			R,I	

Key

S: Statement accesses files with sequential organization.
R: Statement accesses files with relative organization.
I: Statement accesses files with indexed organization.

The OPEN OUTPUT statement opens a file for output. For sequential files, it creates a new file with the designated filename. An existing file with that filename is automatically deleted. For indexed or relative files, use the OUTPUT mode only if the file does not exist. In this case, the file is created. If the file already exists and you want to write to the file, open the file in I-O mode.

The OPEN I-O statement permits the program to open a file for both input and output. I-O mode can be specified only for disk files. For sequential files, OPEN I-O can be specified only for an existing file. For relative or indexed files, the OPEN I-O statement creates the file if it does not already exist.

The EXTEND phrase positions the record pointer immediately after the last logical record of a sequential file, allowing a program to write additional records to that file. EXTEND can be used only for sequential files. Subsequent WRITE statements referencing the file add records as though the file is opened in OUTPUT mode. If the file does not exist, it is created.

The EXCLUSIVE Phrase

The EXCLUSIVE phrase is an Interactive COBOL feature that prevents any other user from opening the file. An attempt to open a file already opened with EXCLUSIVE results in an I/O error.

EXCLUSIVE is redundant for files opened in EXTEND mode, since the system does not allow multiple users to access a file opened for EXTEND. EXCLUSIVE is not implied for files assigned directly to PRINTER or PRINTER-1, whether they are opened for OUTPUT or EXTEND.

Table 8-5 Permissible Options with OPEN

Option	File Is Available	File Is Unavailable
INPUT	Normal open. If the file is empty, the first READ causes the AT END or INVALID KEY condition.	Open is unsuccessful.
I-O	Normal open. If the file is empty, the first READ causes the AT END or INVALID KEY condition.	Sequential: Open is unsuccessful. Indexed or relative: Creates the file.
OUTPUT	Sequential: Deletes old file and creates new one. Indexed or relative: Open is successful.	Creates the file.
EXTEND	Sequential: Positions to last record.	Sequential: Creates the file.

OPEN Rules for Specific Operating Systems

If you do not have the proper access permissions when executing an OPEN statement, the OPEN fails. See your user's guide for information about how the OPEN command works on your system.

In single user mode, you can open as many files as your operating system allows. In multi-user mode, the maximum number of files you can open depends on the total number of files opened on your system.

Open File Limits

Files opened by one or more COBOL programs use operating system input/output file identifiers. An open file identifier is a generic term that refers to the channels, file handles, or file descriptors of your operating system. A sequential file uses a single file identifier for each program that opens the file. An indexed or relative file uses two file identifiers. The number of simultaneously open files depends on the type of file organization and the number of file identifiers available on your operating system. The system limit applies to both single and multi-user mode. If any program attempts to open a file when the system limit is exceeded, the open fails.

PERFORM

Transfers control to one or more procedures.

Unconditional PERFORM Format

```
PERFORM { procedure-name-entry                  }
        { imper-stmt END-PERFORM                }
```

Iterative PERFORM Format

```
        { procedure-name-entry { integer } TIMES                  }
PERFORM {                      { id      }                        }
        { { integer } TIMES imper-stmt END-PERFORM                }
        { { id      }                                             }
```

Conditional PERFORM Format

```
PERFORM { procedure-name-entry UNTIL condition                    }
        { UNTIL condition imper-stmt END-PERFORM                  }
```

Variable Out-of-line PERFORM Format

```
PERFORM procedure-name-entry
     VARYING id-1 FROM id-lit-1 BY id-lit-2 UNTIL condition-1
     [ AFTER id-2 FROM id-lit-3 BY id-lit-4 UNTIL condition-2] ...
```

Variable In-line PERFORM Format

```
PERFORM VARYING id-1 FROM id-lit-1 BY id-lit-2 UNTIL condition-1
     imper-stmt END-PERFORM
```

NOTE: In each of these cases, a *procedure-name-entry* has the following syntax:

```
procedure-name-1 [ { THROUGH } procedure-name-2 ]
                 [ { THRU    }                  ]
```

Examples

```
PERFORM SET-UP THROUGH CALLING-SEQUENCE-EXIT.
PERFORM INCREMENT-RTN 10 TIMES.
PERFORM INCREMENT-RTN UNTIL CHECK-VALUE = QUANTITY.
PERFORM OUTPUT-RTN VARYING SUB-1 FROM CHECK-VALUE BY 2
      UNTIL SUB-1 = 60.
PERFORM
      ADD A TO B
      READ AR-FILE RECORD
END-PERFORM.
PERFORM VARYING I FROM 1 BY 1 UNTIL I > 5
      ADD ITEM(I) TO TOTAL
END-PERFORM.
PERFORM 5 TIMES
      ADD 1 TO I
      DISPLAY ELEMENT(I)
END-PERFORM.
PERFORM UNTIL I=0
      MOVE ELEMENT(I) TO REC1
      WRITE REC1
      SUBTRACT 1 FROM I
END-PERFORM.
```

Rules

When *procedure-name-entry* is included, the PERFORM statement is referred to as an out-of-line PERFORM statement. Thus, Interactive COBOL executes the set of statements contained in each *procedure-name* specified in the PERFORM statement. When *procedure-name-entry* is omitted, the PERFORM statement is referred to as an in-line PERFORM statement. In this case, Interactive COBOL executes only the statements contained within the PERFORM statement itself. To use an in-line PERFORM statement, you must compile your program with Interactive COBOL revision 1.60 or greater.

If you omit *procedure-name-entry* (i.e., it's an in-line PERFORM statement), you must specify *imper–stmt* and the END-PERFORM phrase. The END-PERFORM phrase marks the end of an in-line PERFORM statement. You cannot use the END-PERFORM phrase if you specify *procedure-name-entry* (i.e., an out-of-line PERFORM statement).

You can use the END–PERFORM phrase in Interactive COBOL revision 1.60. (With revisions prior to 1.70, you need to enable END–PERFORM by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

Unless stated otherwise, the same general rules apply to both out–of–line and in–line PERFORM statements.

An *active* PERFORM is one that has begun executing its first statement. The PERFORM remains active until its end limit executes. Interactive COBOL allows 30 active out-of-line and iterative in-line PERFORM statements. An attempt to activate

more than 30 results in a runtime error. The system displays a message and the program terminates.

The words THROUGH and THRU are equivalent.

When you execute an out-of-line PERFORM statement, control passes to the first statement of the procedure named in *procedure-name-1*. Transfer of control to the next executable statement following the PERFORM statement is established as follows:

- If *procedure-name-1* is a paragraph-name and *procedure-name-2* is not specified, the return is after the last statement of the paragraph.
- If *procedure-name-1* is a section-name and *procedure-name-2* is not specified, the return is after the last statement of the last paragraph in the section.
- If *procedure-name-2* is specified and is a paragraph-name, the return is after the last statement of the paragraph.
- If *procedure-name-2* is specified and is a section-name, the return is after the last statement of the last paragraph in the section.
- If control passes to these procedures by means other than a PERFORM statement, control passes through the last statement of the procedure to the next executable statement as if no PERFORM statement mentioned these procedures.

Unconditional PERFORM

The unconditional format is the basic PERFORM statement. The specified set of statements of this PERFORM statement is executed once. Control then passes to the next executable statement following the PERFORM statement.

Iterative PERFORM

The iterative format uses the TIMES option. The initial value of the identifier specifies the number of times the PERFORM executes. If you change this value during execution of the PERFORM, the specified set of statements still executes the original number of times. After the specified set of statements executes the specified number of times, control passes to the next executable statement in the program.

The identifier or integer must be a positive value less than 32,768. If the value is zero or negative when execution of the PERFORM begins, control passes immediately to the next executable statement. If the value of the identifier or integer is greater than 32,767, an error message is displayed and the program terminates.

Conditional PERFORM

The conditional format uses the UNTIL option. The specified set of statements is performed until the condition specified by the UNTIL phrase is true. When the condition is true, control passes to the next executable statement after the PERFORM statement. If the condition is true when the PERFORM statement is encountered, the PERFORM is not executed. Control passes immediately to the next executable statement.

Variable PERFORM

The variable format uses the VARYING option. The identifiers following VARYING and AFTER, and the identifier-literals following BY must be numeric. A variable PERFORM terminates only when *condition-1* is true. *Condition-2* affects only the number of times the PERFORM executes. If you are using code revision 9 or greater, you can have an unlimited number of AFTER clauses, with a different value for *condition–2* in each one. At the end of a PERFORM having two or three conditions, *id-2* and *id-3* have the current values of *id-lit-3* and *id-lit-5*, respectively. Figure 8–1 shows the logic for the PERFORM ... VARYING statement.

NOTE: You cannot specify an AFTER phrase with an in-line PERFORM statement.

When one item is varied, the following occurs:

1. *Id-1* is set to the value specified by *id-lit-1*.
2. The condition of the UNTIL phrase is evaluated.
3. If the condition is true, control passes to the statement following the PERFORM.
4. If the condition is false, the specified set of statements executes once.
5. The value of *id-1* is incremented or decremented by *id-lit-2*.
6. The condition in the UNTIL phrase is tested again. The cycle continues until the condition is true.

When two items are varied, the following occurs:

1. *Id-1* and *id-2* are set to their initial values as specified by *id-lit-1* and *id-lit-3*.
2. *Condition-1* is evaluated.
3. If *condition-1* is true, control passes to the statement following the PERFORM.
4. If *condition-1* is false, *condition-2* is tested.
5. If *condition-2* is false, the PERFORM is executed once, *id-2* is incremented or decremented by *id-lit-4*, and *condition-2* is tested again. This continues until *condition-2* is true.
6. If *condition-2* is true, *id-2* is set to its initial value, *id-1* is incremented or decremented by *id-lit-2*, and control returns to step 2.

When more than two items are varied, the following occurs:

1. *Id-1*, *id-2*, and *id-3* are set to their initial values as specified by *id-lit-1*, *id-lit-3*, and *id-lit-5*.
2. *Condition-1* is evaluated.
3. If *condition-1* is true, control passes to the statement following the PERFORM.
4. If *condition-1* is false, *condition-2* is tested.
5. If *condition-2* is false, the system looks at the next AFTER clause and tests its *condition-2*.
6. If the next *condition-2* is false, the system tests the next *condition–2*. If each *condition–2* in the PERFORM statement is false, the PERFORM is executed, *id-3*

is incremented or decremented by *id-lit-6*, and the condition is tested again until the final *condition-2* becomes true.

7. Once the final *condition-2* is true, *id-3* is set to its initial value, *id-2* is incremented or decremented by *id-lit-4*, and the previous *condition-2* is tested again. This continues until that *condition-2* is true.

8. The system repeats steps 6 and 7 until each *condition–2* is true.

9. If *condition-2* is true, *id-2* is set to its initial value, *id-1* is incremented or decremented by *id-lit-3*, and control returns to step 2.

If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the set of specified statements associated with the included PERFORM must be totally in, or totally excluded from, the logical sequence referred to by the first PERFORM. An active PERFORM statement whose execution point begins within the range of another active PERFORM statement cannot pass to the exit of the first PERFORM statement. Two or more such active PERFORM statements cannot have a common exit. See Figure 8–2 for examples of valid PERFORM constructions.

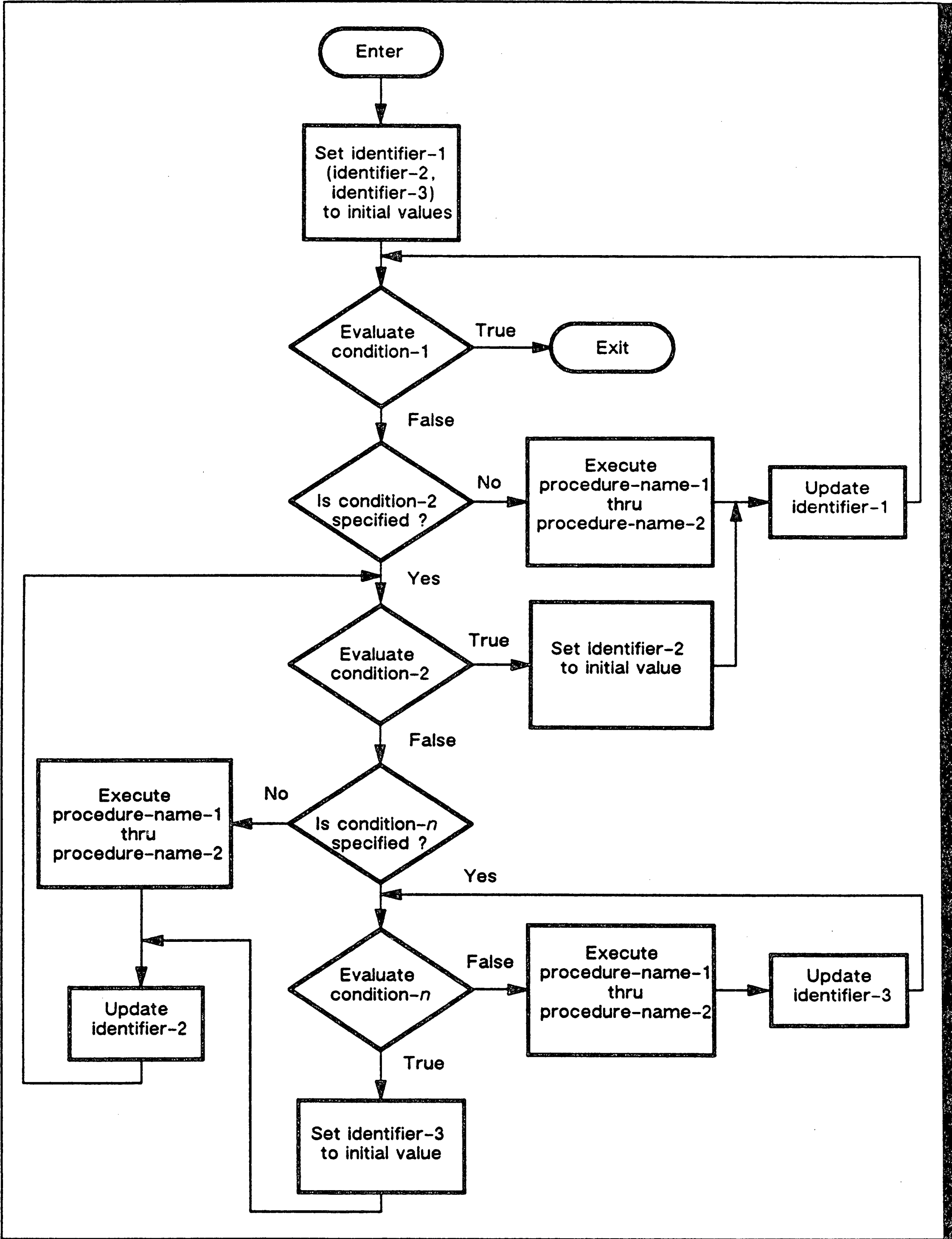


Figure 8-1 Flowchart for a PERFORM ... VARYING Statement

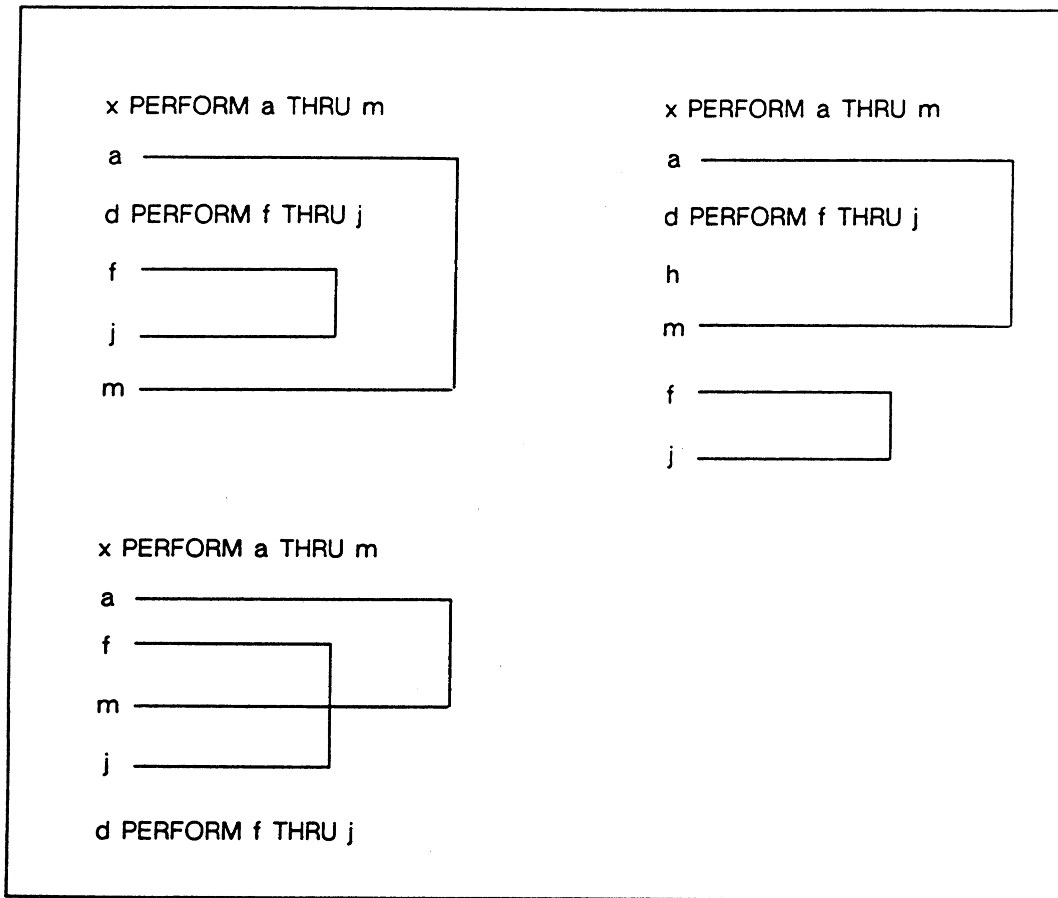


Figure 8–2 Valid PERFORM Constructs

READ

Makes available the next logical record or a specified record, depending on file access type.

Sequential Access Format

READ *filename* RECORD [LOCK] [INTO *id*]
[TIME-OUT AFTER *id-lit*] [; AT END *imper-stmt*] [END-READ]

Random Access Format

Relative Files:

READ *filename* RECORD [LOCK] [INTO *id*] [; INVALID KEY *imper-stmt*]
[END-READ]

Indexed Files:

READ *filename* RECORD [LOCK] [INTO *id*] [; KEY IS *data-name*]
[; INVALID KEY *imper-stmt*] [END-READ]

Dynamic Access Format

Sequential Retrieval:

READ *filename* { NEXT | PREVIOUS } RECORD [LOCK] [INTO *id*] [; AT END *imper-stmt*]
[END-READ]

Random Retrieval:

READ *filename* RECORD [LOCK] [INTO *id*] [; KEY IS *data-name*]
[; INVALID KEY *imper-stmt*] [END-READ]

Examples

```
READ TRANS-FILE INTO REC-2.
READ MASTER-FILE INTO REC-1 INVALID KEY GO TO NO-MAST-RTN.
READ CUSTOMER-FILE NEXT AT END GO TO ENDING-RTN.
READ MASTER-FILE LOCK INTO REC-1
    INVALID KEY GO TO NO-MAST-RETURN.
```

```
05 TIMEVAL PIC S9(3)
...
READ FILE TIME–OUT AFTER TIMEVAL.
```

General Rules

The associated file must be open in the INPUT or I-O mode at the time you execute a READ statement. Execution of a READ statement updates the value of the FILE STATUS item.

The INTO identifier and the record area associated with the filename must not share the same storage area.

When you specify the INTO phrase, the current record is moved from the record area to the identifier area according to the rules for the MOVE statement. Thus, the execution of a READ...INTO statement is equivalent to:

```
READ filename.
MOVE record-name TO identifier.
```

The record is then available in both the input record area and the identifier area. The implied MOVE does not occur if the execution of the READ statement is unsuccessful. Any subscripting or indexing associated with the identifier is evaluated after the record is read and immediately before it is moved to the data-item.

Specify the KEY IS clause only for an indexed file. The data name must be a data-item specified as one of the record keys for the file. The value of any key should not be equal to HIGH-VALUES. This value is reserved for system use.

You can use the END–READ phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the READ statement. (With revisions prior to 1.70, you need to enable END–READ by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

The LOCK Option

Use the LOCK option only for indexed or relative files. When you read a record with the LOCK option, it cannot be accessed by any other user or a subprogram. An attempt by another user to access a record that is locked results in an I/O error. A locked record can be deleted, rewritten, or unlocked only by the program that issued the LOCK statement. An exception to this rule is that after a successful execution of an OPEN INPUT statement, records in that file can be read whether they are locked or unlocked.

When a program attempts a sequential READ of a locked record, and the READ is unsuccessful, the current record pointer is not reset. If the read statement is within a

loop, the program continues to try to access that record until it is unlocked. You can avoid this looping condition by executing a statement to check File Status code 94. File Status code 9E is a result of an unsuccessful lock, not an unsuccessful read. Therefore, the current record pointer is reset.

AT END Condition

A READ statement must specify an AT END or INVALID KEY phrase (depending on the access method) if the program does not provide a USE procedure for the file.

When you execute a sequential READ statement and no next logical record exists in the file, the AT END condition occurs, and the READ statement is unsuccessful. When the AT END condition is recognized, the following actions occur:

1. A value is placed into the FILE STATUS item associated with this file, to indicate an AT END condition.
2. If the AT END phrase is specified in the statement, control passes to the AT END imperative statement. Any USE procedure specified for this file is not executed.
3. If the AT END phrase is not specified, a USE procedure must be specified and executed for this file.

Access Modes

Interactive COBOL provides three methods for retrieving data and three types of file organization (see Chapter 2). The user defines the access method and file organization with the SELECT clause in the File-Control entry. The following sections discuss record retrieval.

Rules for Sequential Access

The sequential access method can be used for files with sequential, relative, or indexed organization. It must be used when the file organization is sequential.

Execution of a sequential READ statement makes available the next logical sequential record from the file. The current record pointer determines the next record.

The TIME-OUT AFTER *id-lit* phrase enables a local time-out. The file you specify must be a tty device; the TIME-OUT feature does not work with any other kind of file. This phrase specifies the amount of time in seconds that Interactive COBOL will wait between keystrokes as you enter the field; it does not specify the amount of time that you have to enter the entire field. The value for *id-lit* must be from 1 through 65535. If the value is 0 or a negative number, an error occurs and the time-out is disabled. Any value greater than 65535 causes the time-out to default to 65535. When a time-out occurs, Interactive COBOL places a value of 9T in the File Status code and the READ fails. The TIME-OUT phrase is available only with code revision 9 or greater.

Sequential Organization: If an OPEN statement positioned the current record pointer, the current record is made available. If a previous READ statement positioned the current record pointer, the next existing record in the file is made available. The next record is the succeeding record in logical sequence.

When the AT END condition occurs, the program must not execute a sequential READ for that file before executing a successful CLOSE statement followed by a successful OPEN statement.

Relative Organization: If a START or OPEN statement positioned the current record pointer, the current record is made available. If a previous READ statement positioned the current record pointer, the next existing record (that is, one that has not been logically deleted) is made available. The next record is the succeeding logical record in key sequence. The key sequence is determined by the ascending values of relative record numbers for the records in the file.

If the RELATIVE KEY clause is specified in the SELECT entry for this file, the relative key data-item is updated to reflect the relative record number of the record being made available.

The AT END condition occurs when no next logical record exists in the file. When this condition is recognized, a sequential access READ statement must not be executed for this file until one of the following has been executed:

- A successful CLOSE statement followed by a successful OPEN statement
- A successful START statement for this file
- A successful random access READ statement for this file

Indexed Organization: If a START or OPEN statement positioned the current record pointer, the current record is made available. If a previous READ statement positioned the current record pointer, the next existing record in the file is made available. The next existing record is the succeeding logical record in key sequence. The key sequence is determined by the ascending values of the current key of reference.

When an ALTERNATE RECORD KEY is the key of reference, file records with duplicate key values are made available in the order in which they were written to the file.

The AT END condition occurs when no next logical record exists in the file. When the AT END condition is recognized, a sequential access READ statement must not be executed for this file until one of the following has been executed:

- A successful CLOSE statement followed by a successful OPEN statement
- A successful START statement for this file
- A successful random access READ statement for this file

Rules for Random Access

Use this method only with disk files having indexed or relative organization. The SELECT clause must specify random access for the file. The current record is determined by the value in the current record pointer.

You must specify the INVALID KEY phrase if no applicable USE procedure is designated for the filename.

Relative Organization: Execution of a random READ statement for a relative file sets the current record pointer to the record whose relative record number is contained in

the RELATIVE KEY data-item (defined in the SELECT clause). That record is then made available. When the record is not found, the INVALID KEY condition exists and execution of the READ statement is unsuccessful. Following an unsuccessful READ, the contents of the associated record area and the position of the current record pointer are undefined.

Indexed Organization: Records in an indexed file are accessed by record key values. When you execute a random READ statement for an indexed file, the following occurs:

1. The key of reference is established. When the KEY phrase is specified, the data-name becomes the key of reference. The KEY data-name must identify a record key associated with filename. It can be the primary record key or any alternate record key. The data-name can be qualified, but it cannot be subscripted or indexed. If the KEY phrase is omitted, the primary record key is established as the key of reference.

2. The value of the key of reference is compared with the value of the corresponding key data-item in the file records, until the first record having an equal value is found.

3. The current record pointer is then set to this record, and it is made available.

If a record with an equal value is not found, the INVALID KEY condition exists and execution of the READ statement is unsuccessful. Following an unsuccessful READ, the contents of the associated record area and the position of the current record pointer are undefined.

Rules for Dynamic Access

This method combines the sequential and random access methods and allows the program to switch from one to the other. This access mode can be used only with files having relative or indexed organizations. The SELECT clause must specify dynamic access for the file.

Sequential Retrieval: The NEXT or PREVIOUS phrase must be specified to retrieve records sequentially during dynamic access. The NEXT phrase causes the program to retrieve the next logical record. Similarly, the PREVIOUS phrase causes the program to retrieve the previous logical record. If the preceding operation was a START or OPEN, then either a READ NEXT or a READ PREVIOUS retrieves the current record. Otherwise, the next or previous record is read. If the end of the file is encountered during a READ NEXT, or if the beginning of the file is encountered during a READ PREVIOUS, the AT END condition occurs and the statement is unsuccessful. The rules outlined for the AT END condition apply.

All the other rules described previously for sequential access apply.

Random Retrieval: The key of reference is established as described previously for Random Access. Subsequent sequential READ statements use this key of reference until a different key is established.

The same rules described previously for random access apply.

The following example shows dynamic access for an indexed file:

```
ENVIRONMENT DIVISION.
...
FILE-CONTROL.
    SELECT CUSTOMER-FILE;
        ASSIGN TO DISK, CUST-FILE-NAME;
        ORGANIZATION IS INDEXED;
        ACCESS IS DYNAMIC;
        RECORD KEY IS CUST-KEY;
        ALTERNATE RECORD KEY IS CUST-AREA.
...
PROCEDURE DIVISION.
...
```

Random read on key value of CUST-KEY

```
READ CUSTOMER-FILE INTO REC-1;
            INVALID KEY MOVE 0 TO REC-EXISTS-FLAG.
```

Sequential read of NEXT record

```
READ CUSTOMER-FILE NEXT INTO REC-2;
            AT END MOVE 1 TO END-OF-CUST-FILE.
```

Record Pointers

Record pointers are updated after you execute a READ statement. Each program maintains its own set of record pointers and does not keep track of other programs' record pointers.

For example, PGM A writes three records and calls PGM B. PGM B reads the three records in its input file. Since the record pointer is updated after the READ statement, PGM B's record pointer is set to end of file. Control returns to PGM A, which writes three more records and calls PGM B. PGM B is unable to execute a READ statement to access the new records written by PGM A, since the record pointer for PGM B is set to end of file. PGM B must execute a START *filename* KEY GREATER THAN *current-key* statement to retrieve the new records.

REWRITE

Replaces an existing record in a disk file.

Format for Sequential Files

REWRITE *record-name* [IMMEDIATE] [FROM *id*] [END-REWRITE]

Format for Relative and Indexed Files

REWRITE *record-name* [IMMEDIATE][FROM *id*][; INVALID KEY *imper-stmt*]
[END-REWRITE]

Examples

```
REWRITE MAST-REC FROM REC-1 INVALID KEY GO TO REC-LOST-RTN.
REWRITE TRANS-REC.
REWRITE CUSTOMER-REC INVALID KEY GO TO ERROR-6.
```

Rules

Record-name is the name of a logical record associated with a file declared in the Data Division. *Record-name* can be qualified. *Record-name* and *id* must not refer to the same storage area.

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.

The file associated with *record-name* must be open in I-O mode at the time you execute the REWRITE statement.

Execution of the REWRITE statement causes the value of the FILE STATUS item to be updated. It does not affect the current record pointer.

After the successful execution of a REWRITE statement, the record is still available in *record-name* unless the associated file is named in a SAME RECORD AREA clause. In this case, the record can be overwritten by a record from another file named in the SAME RECORD AREA clause.

The IMMEDIATE option is available only with code revision 7 or greater. IMMEDIATE is a reserved word that tells Interactive COBOL to write the data (and keys) to disk before the REWRITE statement completes execution. Normally the data is held in an internal buffer before being written to disk. The IMMEDIATE option provides increased file security at the expense of performance.

Execution of a REWRITE statement with the FROM phrase is equivalent to:

```
MOVE id TO record-name.
REWRITE record-name.
```

After the record is rewritten, the record information is still available in the identifier.

The key value of HIGH-VALUES is reserved for system use. Your program should not attempt to execute REWRITE on a record with key values equal to HIGH-VALUES.

For files in sequential access mode, a successful READ statement must be executed immediately before the execution of the REWRITE statement. The record specified in the REWRITE statement replaces the record that was just read.

You can use the END-REWRITE phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the REWRITE statement. (With revisions prior to 1.70, you need to enable END-REWRITE by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

Sequential Files

You cannot specify the INVALID KEY phrase for a file with sequential organization.

Relative Files

In a relative file accessed with random or dynamic access mode, the record specified in the REWRITE statement replaces the record designated by the RELATIVE KEY data-item.

You must specify an INVALID KEY clause or a USE procedure. The INVALID KEY condition occurs when the referenced file does not contain the record specified by the RELATIVE KEY data-item. The file is not updated, and the data in the record is unaffected.

Indexed Files

With sequential access, the record specified in the REWRITE statement replaces the record specified by the value of the primary record key. The value specified in the REWRITE statement must equal the value of the primary key data-item in the record just read from this file.

With random or dynamic access, the record specified in the REWRITE statement replaces the record specified by the value in the primary record key.

The contents of alternate record key data-items of the record being rewritten can differ from those in the record being replaced. If any alternate data-item has changed in value, the record is no longer available using the old alternate values. It is available using the new alternate values. The system ensures that later access to the record can be based upon any of the record keys.

You must specify an INVALID KEY clause or a USE procedure. The INVALID KEY condition exists if either of the following occurs:

- The access mode is sequential, and the value in the primary RECORD KEY of the record to be replaced does not equal the primary RECORD KEY data-item of the last record retrieved from the file.
- The value in the primary RECORD KEY does not equal that of any record in the file.

If either condition exists, the REWRITE statement is unsuccessful, the file is not updated, and the data is unaffected.

SEARCH

Scans a table for a particular table element that satisfies specified conditions.

Format 1

```
SEARCH id-1 [VARYING {id-2          }] [ AT END imper-stmt-1]
                     {index-name-1  }

        { WHEN condition-1 {imper-stmt-2  } } ... [END-SEARCH]
        {                  {NEXT SENTENCE } }
```

Format 2

```
SEARCH ALL id-1 [AT END imper-stmt-1]

       WHEN { data-name-1 {IS EQUAL TO} {id-lit-1               } }
            {             {IS =       } {arithmetic-expression-1} }
            { condition-name-1                                    }

     [ AND { data-name-2 {IS EQUAL TO} {id-lit-2               } } ] ...
     [     {             {IS =       } {arithmetic-expression-2} } ]
     [     { condition-name-2                                    } ]

     {imper-stmt-2  }  [END-SEARCH]
     {NEXT SENTENCE }
```

Examples

```
...
C100-SEARCH.
  MOVE ZERO TO RATE.
  SET RATE-NDX TO 1
  SET JOB-NDX TO 1
  SEARCH JOB-ENTRY VARYING RATE-INDX
    AT END MOVE "JOB # NOT IN TABLE" TO ERR-MESS-PR
    WHEN JOB-IN = JOB-ENTRY(JOB-NDX)
      MOVE RATE-ENTRY(RATE-INDX) TO RATE.
...
01 JOB-TABLE
```

```
02 JOB-ENTRY OCCURS 23 TIMES
             ASCENDING KEY JOB-NUM, JOB-NAME
             INDEXED BY JOB-NDX.
    03 JOB-NUM   PIC 99.
    03  JOB-NAME PIC X(4).
    03 JOB-RATE  PIC 99.

SEARCH ALL JOB-ENTRY
     AT END MOVE "JOB# NOT IN TABLE" TO ERR-MESS-PR;
     WHEN JOB-NUM(JOB-NDX) = JOB-IN AND
          JOB-NAME(JOB-NDX) = NAME-IN
          MOVE JOB-RATE(JOB-NDX) TO RATE-OUT;
END-SEARCH.
```

Rules

The SEARCH statement lets you search a table for an element that satisfies the condition you specify. If you use format 1 of the SEARCH statement, the system searches serially for the element starting at the index-setting you specify. With format 2, the system performs a non-serial search and tries to find an element that satisfies all of the conditions in the WHEN clause.

NOTE: The SEARCH statement is effective only with code revision 9 or greater.

Id-1 must have an OCCURS clause and an INDEXED BY phrase; *id-1* cannot be subscripted. In format 2, the description of *id-1* must also contain the KEY IS phrase in its OCCURS clause.

Id-2 must refer to a data item described as USAGE IS INDEX or as an elementary numeric integer. *Id-2* cannot be subscripted by the first index name specified in the INDEXED BY phrase in the OCCURS clause associated with *id-1* (i.e., the primary index). *Id-2* is incremented each time the primary index is incremented. If you specify an integer value for *id-2*, the system increments *id-2* by 1. Unless the initial value of *id-2* matches the initial value of the primary index when SEARCH is executed, the values of *id-2* and the primary index will not necessarily coincide when SEARCH finishes executing.

In format 1, *condition-1* may be any conditional expression.

In format 2, the value for *condition-name* must be a key and it must be a single value. Include the *data-name* associated with a *condition-name* in the KEY IS clause of *id-1*. All *data-names* can be qualified. You must include each data-name in the KEY IS clause of *id-1;* each data-name must be indexed by the primary index of *id-1* along with any other indices or literals that are required. No identifier in the WHEN clause (*id-lit-1*, *id-lit-2*, or *arithmetic-expression*) can appear in the KEY IS clause of *id-1* or can be indexed by the primary index of *id-1*.

In format 2, when the program refers to a *data-name* in *id-1*'s KEY IS clause, or when the program refers to a *condition-name* associated with a *data-name* in *id-1*'s KEY IS clause, it must also refer to all preceding *data-names* in the KEY IS clause or to their associated *condition-names*. Thus, if you include the third key, you must also include the first and second keys.

Regardless of the order in which you enter the keys, Interactive COBOL checks them according to their ascending key order.

If you specify the END-SEARCH phrase, you cannot specify the NEXT SENTENCE phrase.

You terminate a SEARCH statement by including:

- An END-SEARCH phrase at the same level of nesting.
- A separator period.
- An ELSE or END-IF phrase associated with a a previous IF statement.

With format 1 of the SEARCH statement, the system starts at the current index setting and begins a serial search operation. Before executing the SEARCH statement, the system checks the value of the *index-name* associated with *id-1*. If the value of *index-name* is outside the range for *id-1* (i.e., it is either less than 1 or larger than the number of elements in the table), the system terminates the SEARCH statement and passes control to the sentence following the SEARCH statement or, if you specified the AT END phrase, to *imper-stmt-1*.

When *index-name* contains a valid value, the SEARCH statement checks the conditions in the order that you entered them. It continues to increment *index-name* and sequentially search the table until it either finds a match or the value of *index-name* exceeds the number of elements in the table. If one of the conditions is satisfied, the system halts the search and executes the imperative statement or NEXT SENTENCE associated with that condition. *Index-name* continues to point to table element that satisfied the condition.

In format 1, if you omit the VARYING phrase, the system uses the first *index-name* specified in *id-1*'s INDEXED BY phrase (i.e., the primary index). No changes are made to any other indices for *id-1*. You can use an index other than the primary index by including the VARYING phrase. When the VARYING phrase is present, the system uses the index name you specify as *index-name-1* as long as that name appears in *id-1*'s INDEXED BY phrase. If this is not the case or if the VARYING *id-2* phrase is specified, the system uses *id-1*'s primary index. Also:

- When *index-name-1* refers to an index in another table, the system uses *id-1*'s primary index. However, each time during the search that the system increments the primary index, it also increments *index-name-1*. Thus, *index-name-1* points to same occurrence number of its table as the primary index does for the *id-1* table.
- When the VARYING *id-2* phrase is used and *id-2* is an index data item, then the system increments *id-2* by the same amount as *id-1*'s primary index. If *id-2* is an integer, the system increments *id-2* by 1 each time it increments the primary index. Unless the values of *id-2* and the primary index correspond at the start of the search, they will not necessarily coincide at the end of the search.

With format 2 of the SEARCH statement, the system performs a non-serial search using only the primary index for *id-1*; it does not change any of *id-1*'s other indices. The system ignores the initial setting of the primary index. The setting for the primary index varies during the search operation so that the entire table is searched; however, the setting always stays within the legal range for the table. During a SEARCH ALL, the system attempts to find a table element that satisfies all of the conditions in the WHEN clause. If no table element satisfies all the conditions, control passes to *imper-stmt-1* of the AT END phrase, if specified, or to the end of the SEARCH statement. Regardless, the final setting of the index is not predictable. If all the conditions are satisfied, the index points to the element that meets all the conditions, and control passes to *imper-stmt-2*, if specified, or to the next executable sentence if the NEXT SENTENCE phrase is specified.

In a format 2 SEARCH statement, the results of the SEARCH ALL operation are predictable only when:

- The data in the table are ordered in the same manner as described in the ASCENDING KEY or DESCENDING KEY clause of *id-1*.
- The key(s) referred to in the WHEN phrase identify a unique table element.

After execution of the *imper-stmt-1* or *imper-stmt-2*, that does not terminate with a GO TO statement, control passes to the end of the SEARCH statement.

If *id-1* is subordinate to a data item that contains an OCCURS clause in its data description, you must define an *index-name* for each dimension of the table. Since only the *index-name* associated with *id-1* is modified, if you want to completely search a multidimensional table, you must include several SEARCH statements. The SEARCH statements must be executed using SET statements prior to each SEARCH to ensure that the *index-names* have appropriate values.

You can use the END-SEARCH phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the SEARCH statement. (With revisions prior to 1.70, you need to enable END-SEARCH by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

SET

Loads specific values into associated index-items.

Format

SET { *id*} ... { TO | UP BY | DOWN BY } *id-lit*

Examples

```
SET SUB-1, SUB-2 TO CHECK-VALUE.
SET SUB-2 UP BY 10.
SET GRADE-1 DOWN BY DEMOTE.
```

Rules

An identifier in a SET statement can refer to an index data-item, an index-name, or an elementary integer. A literal must be numeric or numeric edited and can be signed.

If you use the TO phrase, the literal must be positive, and the value of *id-lit* replaces the current value of *id*.

Index-names are associated with a given table through the INDEXED BY option of the OCCURS clause in a data description entry.

Data-items described as USAGE IS INDEX are not considered to be related to particular tables.

UP BY increases *id* by the value of *id-lit*, while DOWN BY decreases *id* by the value of *id-lit*.

START

Positions the record pointer for sequential retrieval of records.

Format 1

```
                        ┌      ┌ =                           ┐  ┐
                        │      │ EQUAL TO                    │  │
                        │      │ >                           │  │
                        │      │ GREATER THAN                │  │
                        │      │ >=                          │  │
                        │      │ NOT <                       │  │
START filename          │ KEY IS NOT LESS THAN              │  │ data-name
                        │      │ <                           │  │
                        │      │ LESS THAN                   │  │
                        │      │ <=                          │  │
                        │      │ LESS THAN OR EQUAL TO       │  │
                        │      │ NOT >                       │  │
                        └      └ NOT GREATER THAN            ┘  ┘

      [; INVALID KEY imper-stmt] [ END-START ]
```

Format 2

```
START filename { FIRST }  [ KEY IS data-name ]
               { LAST  }
      [; INVALID KEY imper-stmt] [ END-START ]
```

Examples

```
START TRANS-FILE KEY IS EQUAL TO FIRST-VALUE.
START INVENTORY-01 KEY IS NOT LESS THAN LAST-VALUE;
    INVALID KEY PERFORM NON-REC-RTN.
START INVENTORY FIRST.
```

Rules

The START statement sets the current record pointer to the first record that satisfies the KEY condition. The filename must be the name of an indexed or relative file with sequential or dynamic access. The file must be open in INPUT or I-O mode. Execution of the START statement updates the value of the FILE STATUS item.

NOTE: The START options LESS, LESS THAN OR EQUAL TO, NOT >, NOT GREATER THAN, <=, FIRST, and LAST work only with code revision 9 or greater.

If you omit the KEY phrase and don't include either the keyword FIRST or LAST, the relational operator IS EQUAL TO is implied. When you use the KEY phrase, the comparison specified in the KEY relational operator is made between the key field associated with the file's records and the corresponding data-name. The specified data-name can be qualified, but it cannot be subscripted or indexed. The value of the key cannot be equal to HIGH-VALUES. HIGH-VALUES is reserved for system use.

You can use the keyword FIRST to indicate that the beginning position for the file is the first record in the files. The keyword LAST indicates that the file pointer will be positioned on the last record in the file. Execution of the next READ then retrieves the last record in the file.

The START statement makes a comparison between a key field in the file's records and the current value in the corresponding key data-name. It then positions the current record pointer to the first undeleted logical record in the file with the key value that satisfies the condition.

If no record in the file satisfies the condition, an INVALID KEY condition exists. The position of the current record pointer is undefined, and INVALID KEY *imper-stmt*, if specified, is executed. If none is specified, the applicable USE procedure is executed. The INVALID KEY phrase must be specified if no applicable USE procedure is designated for the file.

You can use the END–START phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the START statement. (With revisions prior to 1.70, you need to enable END–START by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

Relative Files

For relative files, the data-name must be the data-item specified in the RELATIVE KEY phrase of the associated FILE-CONTROL entry.

If you do not specify the KEY phrase, START uses the RELATIVE KEY data-item and the IS EQUAL TO operation for the comparison. It positions the current record pointer to the first undeleted logical record with the key that satisfies the condition.

If the START statement is unsuccessful, the current record pointer is undefined.

Indexed Files

For indexed files, the data-name is an alphanumeric data-item that specifies the primary key or an alternate record key.

When the statement executes successfully, the key of reference is established. The data-name is the key of reference unless the KEY phrase is not specified, in which case the primary key becomes the key of reference. Subsequent sequential READ statements also use it.

If the operands in the comparison are of unequal length, the comparison proceeds based on the length of the key, without regard to the length of the defined data-name. All other nonnumeric comparison rules apply.

If the START statement specifies the EQUAL phrase, the current record pointer is set to the first record with a key equal to the contents of the data-name. If you use the GREATER phrase, the current record pointer is set to the first record with a key greater than the data-name. If you specify the NOT LESS THAN phrase, the current record pointer is set to the first record with a key greater than or equal to the data-name.

If you execute START EQUAL for a value in a duplicate alternate key, the current record pointer is positioned to the first record written using the alternate key value. Subsequent sequential reads retrieve records in the order in which they were written.

If START is unsuccessful, the current record pointer and the current key of reference are undefined.

STOP

Permanently or temporarily suspends program execution.

Format

```
STOP { RUN     }
     { literal }
```

Examples

```
STOP RUN.
STOP "PROGRAM NOW TERMINATING".
```

Rules

A STOP RUN statement terminates the program. When STOP RUN appears in a consecutive sequence of imperative statements within a sentence, it must be the last statement in that sequence.

The literal can be numeric, nonnumeric, or any figurative constant except ALL. When you execute STOP *literal*, the literal is displayed on the screen and the program halts. Information displayed on the screen is unaffected.

The program remains halted until you press New Line or CR. Execution then continues with the next statement. If you execute the program from the operating system prompt, pressing Esc terminates the program and returns you to the prompt. If you execute the program from Logon, pressing Esc terminates the program and displays a STOP RUN message. Control then returns to Logon when you press New Line, CR, or Esc.

If a file is in OPEN mode when a STOP RUN is executed, the system closes all open files in the run unit—not just those opened by the current program. STOP *literal* does not close open files unless you press the Esc key.

STRING

Lets you place the partial or complete contents of one or more data items into a single data item.

Format

STRING { *id-lit-1* ... DELIMITED BY { *id-lit-2* / SIZE } } ...

INTO *id-3* [WITH POINTER *id-4*]

[ON OVERFLOW *imper-stmt-1*] [END-STRING]

Examples

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DES PIC X(26)
    VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
01 SOURCES.
    02 S1 PIC X(4) VALUE "0123".
    02 S2 PIC X(4) VALUE "4567".
    02 S3 PIC X(6) VALUE ALL "89".
    02 S4 PIC X(4) VALUE "#! &".
    02 S5 PIC X(5) VALUE SPACES.
    02 S6 PIC X(7) VALUE "@%$".
    02 S7 PIC X(6) VALUE "ABCABC".
    02 S8 PIC X(5) VALUE "AAAAA".
    02 S9 PIC X(3) VALUE "333".
    02 S10 PIC X(3) VALUE "EXD".
01 ONEC PIC X VALUE "C".
01 THREE PIC X VALUE "3".
01 EXEX PIC XX VALUE "XX".
01 DEL PIC XX VALUE "12".
01 P PIC 99 VALUE 1.
PROCEDURE DIVISION.
    STRING S1,S2 DELIMITED BY DEL.
         S3 DELIMITED BY SIZE,
```

```
          S4, S5, S6 DELIMITED BY SPACES,
          S7 DELIMIETD BY ONEC,
          S8, S9 DELIMITED BY THREE,
          S10 DELIMITED BY EXEX,
              INTO DES, POINTER P,
                ON OVERFLOW DISPLAY "OVERFLOW".
          DISPLAY "DES= " DES.
          DISPLAY "P= ´ P.
...
DES = 04567898989#!@%$ABAAAAAEXD
P = 27
```

Rules

In the STRING statement, *id-lit-1* represents the sending item, *id-lit-2* or SIZE specifies a string delimiter for this item, *id-3* is the receiving item, and *id-4* specifies the character position in *id-1* at where the STRING operation begins. A single STRING statement can have multiple sending items as well as multiple sending items and delimiting phrases.

NOTE: The STRING statement works only with code revision 9 or greater.

If you include the SIZE phrase in your statement, the system moves the entire content of *id-lit-1* to *id-3*. If you use a figurative constant as *id-lit-2*, it will be interpreted as a single-character, non-numeric literal. If you use a figurative constant for *id-lit-1*, it refers to an implicit one-character data item described as USAGE IS DISPLAY. Do not use a figurative constant that begins with the word ALL for either *id-lit-1* or *id-lit-2*.

All literals used in a STRING statement must be nonnumeric literals. Describe all the identifiers except *id-4* either implicitly or explicitly as USAGE IS DISPLAY.

When *id-lit-1* or *id-lit-2* is an elementary numeric data item, describe it as an integer with the symbol P in its PICTURE character-string.

Id-3 cannot represent an edited data item, or described with the JUSTIFIED clause.

Id-4 is an elementary numeric integer data item. It must be large enough to contain a value that is equal to the size of *id-3* plus 1. Do not use the symbol P in the PICTURE character-string of *id-4*. If you include the POINTER phrase, then the value of *id-4* must always be greater than 0 when the STRING statement begins executing.

As it transfers data from *id-lit-1* to *id-3*, the system follows the rules for alphanumeric to alphanumeric moves, except that it does not provide space fillings.

The system transfers data from *id-lit-1* from left to right. When you have multiple sending items, the system begins with the leftmost *id-lit-1*. If you included *id-lit-2* as your delimiter, the system transfers data from *id-lit-1* until it encounters the characters specified by *id-lit-2* or reaches the end of *id-lit-1*. Then it moves to the next sending item and begins transferring data from that *id-lit-1*. The system does not transfer the characters specified by *id-lit-2*. If you included the SIZE phrase, the

system transfers all the data from the first (leftmost) sending item and then begins transferring data from the next sending item. Regardless of whether you included *id-lit-2* or the SIZE phrase, the system continues to transfer data until either all occurrences of *id-lit-1* have been processed or until it reaches the end of *id-3*.

The system transfers the data from *id-lit-1* to *id-3* one character at a time. When you include the POINTER phrase, the STRING statement uses the value of *id-4* to determine where to place each character it moves to *id-3*. Before it moves a character from *id-lit-1*, the system checks the value of *id-4*. It then places the *id-lit-1* character in *id-3* in the place specified by *id-4*, increments the value of *id-4* by 1, and repeats the process. If you don't include the POINTER phrase, the system responds as if you specified *id-4* with an initial value of 1.

If the value of *id-4* is ever less than 1 or greater than the number of character positions in *id-3*, control transfers to either the end of STRING statement or, if the ON OVERFLOW phrase is specified, to *imper-stmt-1*.

The STRING statement does not clear the existing data from *id-3* before it starts executing. Thus, the system only replaces the portion of *id-3* that is referred to during execution of the statement; the rest of *id-3* will contain data that was present before the STRING began executing.

If *id-lit-1* or *id-lit-2* occupies the same storage area as *id-3* or *id-4*, or if *id-3* and *id-4* occupy the same storage area, the results of the STRING statement will be undefined.

You can use the END-STRING phrase in Interactive COBOL revision 1.70 or greater to terminate the scope of the STRING statement.

SUBTRACT

Subtracts one or the sum of two or more numeric data-items from an item, and sets the value of an item equal to the result.

Overlaying SUBTRACT Format

SUBTRACT { *id-lit*}... FROM { *id* [ROUNDED]} ...
[; ON SIZE ERROR *imper-stmt*] [END-SUBTRACT]

Nonoverlaying SUBTRACT Format

SUBTRACT { *id-lit*}... FROM *id-lit* GIVING { *id* [ROUNDED]} ...
[; ON SIZE ERROR *imper-stmt*] [END-SUBTRACT]

Examples

```
SUBTRACT PAYMENT FROM BALANCE;
    ON SIZE ERROR PERFORM BAL-ERR-RTN.
SUBTRACT DEDUCTIONS FROM GROSS GIVING NET;
    ON SIZE ERROR PERFORM NET-ERR-RTN.
```

Rules

Each *id* must refer to an elementary numeric item. However, in nonoverlaying format each *id* following the word GIVING can refer to an elementary numeric or numeric edited item. (With code revision 8, you can have multiple destinations for an arithmetic statement.)

Each literal must be a numeric literal.

In the overlaying format, all literals or identifiers preceding the word FROM are added together, this total is subtracted from the current value of *id*, and the result is stored in *id*. Interactive COBOL repeats this process for each *id* that follows the word FROM. For example, in the sentence:

```
SUBTRACT A, B FROM C, D.
```

the sum A + B is subtracted from C, and the result is stored in C. Then the sum of A + B is subtracted from D and that result is stored in D.

In the nonoverlaying format, all literals or identifiers preceding the word FROM are added together, the sum is subtracted from the *id-lit* following the word FROM, and the result of the subtraction is stored as the value of each *id* specified in the GIVING phrase. For example, in the sentence:

```
SUBTRACT A, B FROM C GIVING D, F.
```

the sum A + B is subtracted from C, the result is stored in D and in F, and the values in A, B, and C are unchanged.

You can use the END-SUBTRACT phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the SUBTRACT statement. (With revisions prior to 1.70, you need to enable END-SUBTRACT by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

SUBTRACT CORRESPONDING

Subtracts items in one group from items in another group that have the same name.

Format

SUBTRACT { CORRESPONDING / CORR } *id* FROM *id* [ROUNDED]

[; ON SIZE ERROR *imper-stmt*] [END-SUBTRACT]

Examples

```
SUBTRACT CORRESPONDING LIST1 FROM LIST2.
SUBTRACT CORR INVEN-SOLD FROM CURR-INVEN;
    ON SIZE ERROR GO TO X3.
```

Rules

Both identifiers must refer to group items. The elements within the groups must all be elementary items and none may be designated as FILLER, REDEFINES, OCCURS, or USAGE IS INDEX. Level 77 or level 88 items cannot be used.

The groups do not need to contain exactly the same list of ids. SUBTRACT CORRESPONDING compares the elements which make up each group. When two items are found that have the same name, the first operand is subtracted from the second, and the result is stored in the second operand. Those elements with different names are ignored.

If a SIZE ERROR occurs on any one subtraction, all of the remaining subtractions are performed before the imperative statement executes.

The following examples illustrate the use of SUBTRACT CORRESPONDING:

```
01  LIST1.                          01  LIST2.
    02  APPLES      PIC 99.             02  LEMONS      PIC 99.
    02  ORANGES     PIC 99.             02  APPLES      PIC 99.
    02  PEARS       PIC 99.             02  ORANGES     PIC 99.
```

The initial values are:

```
LIST1                               LIST2
    APPLES      4                       LEMONS     10
    ORANGES     2                       APPLES      8
    PEARS       5                       ORANGES     6
```

After execution of SUBTRACT CORRESPONDING LIST1 FROM LIST2 the values are:

```
LIST1                              LIST2
    APPLES      4                      LEMONS   10
    ORANGES     2                      APPLES    4
    PEARS       5                      ORANGES   4
```

The CORRESPONDING phrase does not require that level numbers be identical. However, the record structures must be the same. For example:

```
01  LIST1.                         01  LIST2.
    02  APPLES      PIC 99.            02  SUBLIST2.
    02  ORANGES     PIC 99.                03  LEMONS      PIC 99.
    02  PEARS       PIC 99.                03  APPLES      PIC 99.
                                           03  ORANGES     PIC 99.
```

The statement SUBTRACT CORRESPONDING LIST1 FROM LIST2 is illegal. The statement SUBTRACT CORRESPONDING LIST1 FROM SUBLIST2 is legal.

You can use the END-SUBTRACT phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the SUBTRACT CORRESPONDING statement. (With revisions prior to 1.70, you need to enable END-SUBTRACT by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

UNDELETE

Restores a logically deleted record to an indexed or relative file.

Format

UNDELETE *filename* RECORD [; INVALID KEY *imper-stmt*]
[END-UNDELETE]

Examples

```
UNDELETE CUSTOMER-FILE RECORD INVALID KEY GO TO NO-REC-RTN .
UNDELETE INVENTORY-01.
UNDELETE MASTER-FILE INVALID KEY GO TO ERROR-11.
```

Rules

The system restores to a file the logically deleted record that is identified by the contents of the RECORD KEY or RELATIVE KEY data-item associated with the specified filename. After the successful execution of an UNDELETE statement, the record can be accessed as before the deletion.

The INVALID KEY phrase must be specified when no applicable USE statement has been defined for the file. The INVALID KEY condition exists when the file does not contain the record specified or the record specified is not deleted.

Execution of an UNDELETE statement does not affect the contents of the record area associated with the filename.

An UNDELETE statement updates the FILE STATUS item. The current record pointer is not affected.

The file associated with *filename* must be open in I-O or OUTPUT mode at the time you execute the UNDELETE statement.

You can use the END-UNDELETE phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the UNDELETE statement. (With revisions prior to 1.70, you need to enable END-UNDELETE by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

UNLOCK

Releases all records of a specified file that have been locked by READ statements.

Format

<u>UNLOCK</u> *filename* { RECORD / RECORDS }

Example

```
UNLOCK MASTER-FILE RECORDS.
```

Rules

The associated file must be open in INPUT or I-O mode when you execute UNLOCK.

The current record pointer is not affected by an UNLOCK statement.

UNLOCK unlocks only records locked by the READ statements in your program.

UNSTRING

Separates data in a single data item and places it in multiple fields.

Format

UNSTRING *id-1* [DELIMITED BY [ALL] *id-lit-2* [OR [ALL] *id-lit-3*]...]
INTO { *id-4* [DELIMITER IN *id-5*] [COUNT IN *id-6*] }...
[WITH POINTER *id-7*] [TALLYING IN *id-8*]
[ON OVERFLOW *imper-stmt-1*] [END-UNSTRING]

Example

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WORDSS            PIC X(22) VALUE "GONEZWITHZTHEZWINDZ".
01 SOURCES.
   02 S1             PIC X(4).
   02 S2             PIC X(4).
   02 S3             PIC X(3).
   02 S4             PIC X(4).
01 P                 PIC 99 VALUE 1.
01 STOPPER           PIC X VALUE "Z".
PROCEDURE DIVISION.
UNSTRING WORDSS DELIMITED BY STOPPER
  INTO S1 S2 S3 S4 POINTER P
      ON OVERFLOW DISPLAY "OVERFLOW".
```

Rules

In the UNSTRING statement, *id-1* is the data item you want to separate (the sending item), *id-2* and *id-3* are delimiters that specify how to separate this item, *id-4* is the receiving item for the new data item(s), and *id-5* is the receiving area for delimiters. *Id-6* contains the number of characters that are moved from *id-1* to *id-4*, excluding the delimiter characters; *id-7* indicates the position within *id-1* where the character transfer begins. *Id-8* is a counter that is incremented by 1 for each *id-4* modified. A single UNSTRING statement can have multiple receiving items (*id-4*) as well as multiple delimiters.

NOTE: The UNSTRING statement works only with code revision 9 or greater.

The identifiers used for *id-1*, *id-lit-2*, *id-lit-3*, and *id-5* must refer to data items described either implicitly or explicitly as alphanumeric. *Id-4* must be described implicitly or explicitly as USAGE IS DISPLAY. It can be either alphabetic, alphanumeric, or numeric, but you cannot use the symbol P in the PICTURE character-string.

You can include the word ALL in front of the identifier, such as ALL *id*. If you are using literals for *id-lit-2* and *id-lit-3,* they must be non-numeric. If you supply a figurative constant as the delimiter, it stands for a single-character nonnumeric literal. Each *id-lit-2* and *id-lit-3* represents one delimiter and can contain any character in the computer's character set. If a delimiter contains more than one character, all of the characters must be present in that order in the sending item if they are to be recognized as a delimiter.

Id-6 and *id-8* refer to integer data items. Do not use the symbol P in the PICTURE character-string.

Id-7 is an elementary numeric integer data item. If you include the POINTER phrase, the value of *id-7* must always be greater than 0 when the UNSTRING statement begins executing. Also, *id-7* must be large enough to contain a value that is equal to the size of *id-1* plus 1. Do not use the symbol P in the PICTURE character-string of *id-7*.

You can include the DELIMITER IN phrase and/or the COUNT IN phrase only if you include the DELIMITED BY phrase.

The system moves data from *id-1* one character at a time from left to right using the rules for alphanumeric to alphanumeric moves (except that it does not provide space fillings). Unless you use *id-7* to specify which character in *id-1* to move, the system begins with the leftmost character in *id-1*. The system continues to transfer data from *id-1* to *id-4* until (1) it encounters the delimiters specified using *id-lit-2* and *id-lit-3*, (2) it has reached the end of *id-1*, or (3) it has reached the end of *id-4*. If you included the DELIMITED BY phrase and the system fails to find a match for *id-lit-2* or *id-lit-3*, then the UNSTRING operation terminates.

If you include the COUNT IN phrase, the system counts the number of characters it examines in *id-1* that are not delimiter characters and places this number in *id-6*.

If you have multiple receiving areas (*id-4*), once one is full, the system begins moving data to the next one. It continues doing this until either it reaches the end of *id-1* or there are no more receiving areas.

When you include the ALL phrase, the system treats contiguous occurrences of *id-lit-2* or *id-lit-3* as one occurrence; thus, only one occurrence of *id-lit-2* or *id-lit-3* is moved to the receiving data item for delimiters. If the system encounters two contiguous delimiters and ALL is not specified, it fills the receiving area for the second delimiter with spaces if the receiving areas are alphabetic or alphanumeric and with zeros if the receiving areas are numeric.

When you use the DELIMITED BY phrase to specify multiple delimiters, an OR condition exists between them. The system compares each delimiter to the sending field. If a match occurs, the characters in the sending field are considered to be a part of more than one delimiter. The system compares the delimiters to the sending item in the order in which you entered them on the UNSTRING statement.

You can include the DELIMITER IN phrase only if you also include the DELIMITED BY phrase. The DELIMITER IN phrase places the delimiter that stopped the transaction in *id-5*.

If you use the POINTER and TALLYING phrases, then you provide the initial values for *id-7* and *id-8*. The system increments *id-7* by one each time it examines a character in *id-1*. When an UNSTRING statement finishes executing, *id-7* contains the number of characters examined in *id-1* plus the initial value of *id-7*. The system increments *id-8* by 1 for each time *id-4* modified. Thus, when the UNSTRING statement finishes, *id-8* contains the number of times *id-4* was accessed plus the initial value of *id-8*.

An overflow condition occurs if the system initiates UNSTRING and the value *id-7* is less than 1 or greater than the size of *id-1*. This condition also occurs if all the receiving areas are full, but *id-1* still contains characters.

When an overflow condition exists, the UNSTRING operation is terminated and control is transferred to the end of the UNSTRING statement or, if the ON OVERFLOW phrase is specified, to *imper-stmt-1*.

If *id-1* or *id-lit-2* or *id-lit-3* occupies the same storage area as *id-4* or *id-5*, id-6, *id-7*, or *id-8*, or if *id-4*, *id-5*, or *id-6* occupy the same storage area as *id-7* or *id-8*, or if *id-7* and *id-8* occupy the same storage area, the results of the UNSTRING statement will be undefined.

You can use the END-UNSTRING phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the UNSTRING statement. (With revisions prior to 1.70, you need to enable END-UNSTRING by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

USE

In the Declaratives section, specifies procedures for input-output error handling.

Format

USE AFTER STANDARD { EXCEPTION | ERROR } PROCEDURE ON { INPUT | OUTPUT | I-O | EXTEND | { *filename* } ... }

Example

```
PROCEDURE DIVISION.

DECLARATIVES.
PROCESS-ERR SECTION.
    USE AFTER ERROR PROCEDURE ON INPUT.
FIRST-PARAG.
    DISPLAY INPUT-FILE-STAT.
    MOVE 1 TO INPUT-ERROR.
END DECLARATIVES.
```

Rules

A USE statement must immediately follow a section header in the Declaratives section and must be followed by a period and a space. The remainder of the section consists of paragraphs that define the procedures to be used.

The USE statement itself never executes. Instead, it defines the conditions that cause execution of the succeeding procedures. The specified procedures execute when an exceptional I/O condition occurs. After a declarative procedure executes, control passes to the statement following the one that caused the error or exceptional condition.

The words ERROR and EXCEPTION are interchangeable.

The files referenced in a USE statement need not have the same organization or access mode.

The following errors cause USE procedures to be executed:

- One or more filenames are specified, and an error condition occurs during the processing of one of the files.
- The INPUT phrase is specified, and an error condition occurs during the processing of a file open for input.

- The OUTPUT phrase is specified, and an error condition occurs during the processing of a file open for output.
- The I-O phrase is specified, and an error condition occurs during the processing of a file open for I-O.
- The EXTEND phrase is specified, and an error condition occurs during the processing of a file open for extension.
- An AT END or INVALID KEY condition occurs, and the AT END or INVALID KEY phrase has not been specified.

A given filename can be specified in only one USE procedure. This means that if an input file is named in one USE procedure, the program cannot also have a USE procedure that refers to all input files.

A statement within a USE procedure cannot reference a nondeclarative procedure. Likewise, a statement in a nondeclarative procedure cannot reference a procedure-name that appears in the Declaratives section, with one exception: a PERFORM statement in the nondeclarative portion can refer to a USE statement or to the procedures associated with a USE statement. However, the procedures must be contained within one USE statement.

A USE procedure cannot cause the execution of another USE procedure that has been invoked but has not yet returned control to the invoking routine.

WRITE

Releases a logical record to a file.

Format with Sequential Organization

```
WRITE record-name [ FROM id-1]
  [ { BEFORE } ADVANCING { { integer } [ { LINE  } ] } ]  [ END-WRITE ]
  [ { AFTER  }           { { id-2    } [ { LINES } ] } ]
  [                      { PAGE                    } ]
```

```
WRITE record-name [ IMMEDIATE ] [ FROM id ] [ END-WRITE ]
```

Format with Indexed or Relative Organization

```
WRITE record-name [ IMMEDIATE ] [ FROM id] [; INVALID KEY imper-stmt]
    [ END-WRITE ]
```

Examples

```
WRITE TRANS-REC FROM REC-2.
WRITE PRINT-REC-1 BEFORE ADVANCING 2 LINES.
WRITE PRINT-RPT-LN FROM ERROR-RPT AFTER ADVANCING LINE-NUM.
WRITE INVENTORY-REC INVALID KEY GO TO DUPLICATE-REC-RTN.
```

Rules

Execution of the WRITE statement releases a logical record to the file associated with the *record-name*.

The *record-name* is the name of a logical record described in the File Section of the Data Division. The associated file must be open for OUTPUT, I-O, or EXTEND. *Record-name* and *id* cannot reference the same storage area.

After the successful execution of a WRITE statement, the record is still available in *record-name* unless the associated file is named in a SAME RECORD AREA clause. In this case, the record can be overwritten by a record from another file named in the SAME RECORD AREA clause.

The IMMEDIATE option is available only with code revision 7 or greater, and cannot be used with the ADVANCING phrase. IMMEDIATE is a reserved word that tells Interactive COBOL to write the data (and keys) to disk before the WRITE statement completes execution. Normally the data is held in an internal buffer before being written to disk. The IMMEDIATE option provides increased file security at the expense of performance.

The result of the execution of a WRITE with the FROM phrase is equivalent to the statements:

```
MOVE id TO rec-name.
WRITE rec-name.
```

In this case, the record information is still available in *id* after the WRITE.

A WRITE statement updates the FILE STATUS item. The current record pointer is unaffected by a WRITE statement.

The key value of HIGH-VALUES is reserved for system use. Your program should not attempt to write a record with key values equal to HIGH-VALUES.

You can use the END–WRITE phrase in Interactive COBOL revision 1.60 or greater to terminate the scope of the WRITE statement. (With revisions prior to 1.70, you need to enable END–WRITE by compiling your program with the appropriate compiler switch. See your user's guide for information on compiler switches.)

Sequential Organization

The WRITE statement cannot be used in sequential access for a file open in I-O mode.

The ADVANCING phrase controls the vertical positioning of each line on a logical page. An integer must be in the range 0-80. An identifier can have a value up to 65,535. When you omit the ADVANCING phrase, AFTER ADVANCING 1 LINE is implied.

When the ADVANCING phrase is used or implied, the following rules apply:

- BEFORE ADVANCING places the record on the logical page before advancing the current position *n* lines.
- AFTER ADVANCING places the record on the logical page after advancing the current position *n* lines.
- PAGE places the record on the logical page before or after advancing to the next logical page.
- The ADVANCING phrase can be used only with files assigned to PRINTER, PRINTER-1, or DISPLAY.

When you attempt to write beyond the device limits (total available storage capacity) of a sequential file, an exception condition exists and the contents of the record area is unaffected. The following actions take 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 AFTER STANDARD EXCEPTION declarative is specified for the file, that declarative procedure is executed.
- If a USE AFTER STANDARD EXCEPTION declarative is not explicitly or implicitly specified for the file, the result is a fatal I/O error.

Relative Organization

A WRITE statement can be executed for a relative file opened in I-O or OUTPUT mode.

When you open a file in OUTPUT mode, the WRITE statement causes the following actions:

- If the access mode is sequential, the first record released has relative record number 1, the second number 2, the third number 3, and so on. If the RELATIVE KEY data-item is specified in the FILE-CONTROL entry for the associated file, the relative record number of the record just released is placed in the RELATIVE KEY data-item.
- If the access mode is random or dynamic, the RELATIVE KEY data-item must contain the relative record number for this record before the WRITE statement is issued. When the WRITE statement is executed, that record is placed at the specified relative record number position in the file.

When you open a file in I-O mode, the access mode must be random or dynamic. The WRITE statement inserts new records into the file. The value of the RELATIVE KEY data-item must be initialized by the program. Execution of a WRITE statement then places the record at the specified record number position in the file.

The INVALID KEY condition exists when the access mode is random or dynamic, and the RELATIVE KEY data-item specifies a record that already exists in the file.

When the system recognizes the INVALID KEY condition, the WRITE statement is unsuccessful and the FILE STATUS item is updated.

Indexed Organization

When you execute the WRITE statement, the system releases the record, using the contents of the record key and any alternate keys in such a way that subsequent access of the record can be based upon those keys.

Before execution of the WRITE statement, the program must set the data-item specified as the primary record key (the RECORD KEY data-item defined in the FILE-CONTROL entry) to the desired value. The RECORD KEY value must be unique within the file records. Also, you must assign values to the entire record, including all alternate keys and other data fields. Otherwise, the data transferred by a WRITE statement is undefined.

A WRITE statement can be executed for an indexed file opened in OUTPUT or I-O mode.

If you open a file in OUTPUT mode and specify sequential access, records must be released to the system in ascending order of primary record key values.

If you open a file in I-O mode, or specify random or dynamic access, records can be released to the system in any order. Sequential access is not permitted with I-O mode.

When you specify the ALTERNATE KEY clause in the FILE-CONTROL entry, the value of each alternate key is used to construct an alternate path to the data record.

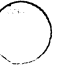
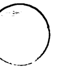
If duplicate records exist, the system stores the records so that sequential access to the file allows retrieval in the order the records were written.

The INVALID KEY phrase must be specified if no applicable USE procedure is designated for this file. An INVALID KEY condition exists if:

- Sequential access is specified for a file opened in OUTPUT mode and the value of the record key is not greater than that of the previous record.
- The file is opened in OUTPUT or I-O mode and the value of the RECORD KEY equals that of an existing record.

When the system recognizes the INVALID KEY condition, execution of the WRITE statement is unsuccessful, and the FILE STATUS item is set to a value indicating the cause of the condition.

End of Chapter

Appendix A
ANSI Standard and Interactive COBOL Reserved Words

This appendix lists ANSI standard and Interactive COBOL reserved words. An asterisk (*) marks the Interactive COBOL reserved words. A tilde (-) marks keywords that are enabled only when you compile your program with the appropriate compiler switch.

ACCEPT
ACCESS
ADD
ADVANCING
AFTER
ALL
ALPHABET
ALPHABETIC
- ALPHABETIC-LOWER
- ALPHABETIC-UPPER
- ALPHANUMERIC
- ALPHANUMERIC-EDITED
- ALSO
- ALTER
ALTERNATE
AND
- ANY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
* AUTO
* BACKGROUND-COLOR
BEFORE
* BELL
BINARY
* BLACK
BLANK
* BLINK
BLOCK
* BLUE
- BOTTOM
* BRIGHT
* BROWN
BY
CALL
CANCEL
- CD
- CF
- CH
CHARACTER
CHARACTERS
- CLASS
- CLOCK-UNITS
CLOSE
- COBOL
- CODE
CODE-SET
* COL
COLLATING
COLUMN
COMMA
- COMMON
- COMMUNICATION
COMP
COMPUTATIONAL
COMPUTE
CONFIGURATION
CONTAINS
- CONTENT
- CONTINUE
- CONTROL
- CONTROLS
- CONVERTING
COPY
CORR
CORRESPONDING
COUNT
CURRENCY
* CYAN
DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
- DAY-OF-WEEK
- DE
- DEBUG-CONTENTS
DEBUGGING
- DEBUG-ITEM
- DEBUG-LINE
- DEBUG-NAME
- DEBUG-SUB-1
- DEBUG-SUB-2
- DEBUG-SUB-3
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
- DESTINATION
DETAIL
* DIM
- DISABLE
* DISK
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC
- EGI
ELSE

EMI
- ENABLE
END
* END-ACCEPT
END-ADD
END-CALL
END-COMPUTE
END-DELETE
* END-DISPLAY
END-DIVIDE
- END-EVALUATE
END-IF
END-MULTIPLY
- END-OF-PAGE
END-PERFORM
END-READ
- END-RECEIVE
- END-RETURN
END-REWRITE
END-SEARCH
END-START
END-STRING
END-SUBTRACT
* END-UNDELETE
END-UNSTRING
END-WRITE
- ENTER
ENVIRONMENT
* EOF
* EOL
- EOP
EQUAL
* ERASE
ERROR
* ESCAPE
- ESI
- EVALUATE
- EVERY
EXCEPTION
* EXCLUSIVE
EXIT
EXTEND
- EXTERNAL
- FALSE
FD
FILE
FILE-CONTROL
FILLER

- FINAL
FIRST
* FIXED
- FOOTING
FOR
* FOREGROUND-COLOR
FROM
* FULL
- GENERATE
GIVING
GO
- GLOBAL
GREATER
* GREEN
- GROUP
- HEADING
* HIGHLIGHT
HIGH-VALUE
HIGH-VALUES
I-O
I-O-CONTROL
IDENTIFICATION
IF
* IMMEDIATE
IN
INDEX
INDEXED
- INDICATE
INITIAL
INITIALIZE
- INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS
JUST
JUSTIFIED
KEY
* KEYBOARD
LABEL
LAST
LEADING
LEFT
- LENGTH
LESS

- LIMIT
- LIMITS
- LINAGE
- LINAGE-COUNTER
LINE
- LINE-COUNTER
LINES
LINKAGE
LOCK
* LOWLIGHT
LOW-VALUE
LOW-VALUES
* MAGENTA
MEMORY
- MERGE
- MESSAGE
MODE
MODULES
MOVE
- MULTIPLE
MULTIPLY
* NAME
NATIVE
NEGATIVE
NEXT
NO
NOT
NUMBER
NUMERIC
- NUMERIC-EDITED
OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
- OPTIONAL
OR
- ORDER
ORGANIZATION
- OTHER
OUTPUT
OVERFLOW
PACKED-DECIMAL
- PADDING
PAGE

- \- PAGE-COUNTER
- PERFORM
- \- PF
- \- PH
- PIC
- PICTURE
- PLUS
- POINTER
- \- POSITION
- POSITIVE
- * PREVIOUS
- * PRINTER
- * PRINTER-1
- \- PRINTING
- PROCEDURE
- \- PROCEDURES
- \- PROCEED
- PROGRAM
- PROGRAM-ID
- \- PURGE
- \- QUEUE
- QUOTE
- QUOTES
- RANDOM
- \- RD
- READ
- \- RECEIVE
- RECORD
- * RECORDING
- RECORDS
- ■ * RED
- REDEFINES
- \- REEL
- \- REFERENCE
- \- REFERENCES
- RELATIVE
- \- RELEASE
- REMAINDER
- \- REMOVAL
- \- RENAMES
- ■ REPLACE
- REPLACING
- \- REPORT
- \- REPORTING
- \- REPORTS
- * REQUIRED
- \- RERUN
- \- RESERVE
- \- RESET
- \- RETURN
- ■ * REVERSE-VIDEO
- \- REVERSED
- \- REWIND
- REWRITE
- \- RF
- \- RH
- RIGHT
- ROUNDED
- RUN
- SAME
- * SCREEN
- \- SD
- SEARCH
- SECTION
- * SECURE
- SECURITY
- \- SEGMENT
- \- SEGMENT-LIMIT
- SELECT
- \- SEND
- SENTENCE
- SEPARATE
- SEQUENCE
- SEQUENTIAL
- SET
- SIGN
- SIZE
- \- SORT
- \- SORT-MERGE
- \- SOURCE
- SOURCE-COMPUTER
- SPACE
- SPACES
- SPECIAL-NAMES
- STANDARD
- STANDARD-1
- \- STANDARD-2
- START
- STATUS
- STOP
- STRING
- \- SUB-QUEUE-1
- \- SUB-QUEUE-2
- \- SUB-QUEUE-3
- SUBTRACT
- \- SUM
- \- SUPPRESS
- * SWITCH
- \- SYMBOLIC
- SYNC
- SYNCHRONIZED
- \- TABLE
- TALLYING
- TAPE
- \- TERMINAL
- \- TERMINATE
- \- TEST
- \- TEXT
- THAN
- THEN
- THROUGH
- THRU
- TIME
- * TIME-OUT
- TIMES
- TO
- \- TOP
- ■ TRUE
- TRAILING
- \- TYPE
- * UNDELETE
- ■ * UNDERLINE
- \- UNIT
- * UNLOCK
- UNSTRING
- UNTIL
- UP
- \- UPON
- USAGE
- USE
- * USER
- USING
- VALUE
- VALUES

* VARIABLE	WITH	ZERO
VARYING	WORDS	ZEROES
WHEN	WORKING-STORAGE	ZEROS
* WHITE	WRITE	

End of Appendix

Appendix B
ASCII Character Sets

Table B-1 ASCII Character Set

Decimal	Octal	Hex	Character	Decimal	Octal	Hex	Character
0	000	00	NUL	38	046	26	&
1	001	01	SOH	39	047	27	'
2	002	02	STX	40	050	28	(
3	003	03	ETX	41	051	29	)
4	004	04	EOT	42	052	2A	*
5	005	05	ENQ	43	053	2B	+
6	006	06	ACK	44	054	2C	,
7	007	07	BEL	45	055	2D	–
8	010	08	BS	46	056	2E	.
9	011	09	HT	47	057	2F	/
10	012	0A	NL	48	060	30	0
11	013	0B	VT	49	061	31	1
12	014	0C	FF	50	062	32	2
13	015	0D	CR	51	063	33	3
14	016	0E	SO	52	064	34	4
15	017	0F	SI	53	065	35	5
16	020	10	DLE	54	066	36	6
17	021	11	DC1	55	067	37	7
18	022	12	DC2	56	070	38	8
19	023	13	DC3	57	071	39	9
20	024	14	DC4	58	072	3A	:
21	025	15	NAK	59	073	3B	;
22	026	16	SYN	60	074	3C	
23	027	17	ETB	61	075	3D	=
24	030	18	CAN	62	076	3E	
25	031	19	EM	63	077	3F	?
26	032	1A	SUB	64	100	40	@
27	033	1B	ESC	65	101	41	A
28	034	1C	FS	66	102	42	B
29	035	1D	GS	67	103	43	C
30	036	1E	RS	68	104	44	D
31	037	1F	US	69	105	45	E
32	040	20	SP	70	106	46	F
33	041	21	!	71	107	47	G
34	042	22	“ ”	72	110	48	H
35	043	23	#	73	111	49	I
36	044	24	$	74	112	4A	J
37	045	25	%	75	113	4B	K

(continues)

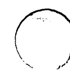
Table B-1 ASCII Character Set

Decimal	Octal	Hex	Character	Decimal	Octal	Hex	Character
76	114	4C	L	102	146	66	f
77	115	4D	M	103	147	67	g
78	116	4E	N	104	150	68	h
79	117	4F	O	105	151	69	i
80	120	50	P	106	152	6A	j
81	121	51	Q	107	153	6B	k
82	122	52	R	108	154	6C	l
83	123	53	S	109	155	6D	m
84	124	54	T	110	156	6E	n
85	125	55	U	111	157	6F	o
86	126	56	V	112	160	70	p
87	127	57	W	113	161	71	q
88	130	58	X	114	162	72	r
89	131	59	Y	115	163	73	s
90	132	5A	Z	116	164	74	t
91	133	5B	[	117	165	75	u
92	134	5C	\	118	166	76	v
93	135	5D	]	119	167	77	w
94	136	5E	^	120	170	78	x
95	137	5F	_	121	171	79	y
96	140	60	‘	122	172	7A	z
97	141	61	a	123	173	7B	{
98	142	62	b	124	174	7C	|
99	143	63	c	125	175	7D	}
100	144	64	d	126	176	7E	-
101	145	65	e	127	177	7F	DEL

(concluded)

Table B-2 DG International Symbols (8-bit ASCII character set)

Decimal	Octal	Hex	Character	Decimal	Octal	Hex	Character
160	240	A0	space	212	324	D4	Ö
161	241	A1	space	213	325	D5	Õ
162	242	A2	space	214	326	D6	Ø
163	243	A3	space	215	327	D7	Œ
164	244	A4	space	216	330	D8	Ú
165	245	A5	space	217	331	D9	Ù
166	246	A6	¤	218	332	DA	Û
167	247	A7	¢	219	333	DB	Ü
168	250	A8	£	220	334	DC	space
169	251	A9	space	221	335	DD	space
170	252	AA	space	222	336	DE	space
171	253	AB	!	223	337	DF	space
172	254	AC	¿	224	340	E0	á
173	255	AD	space	225	341	E1	à
...	...			226	342	E2	â
185	271	B9	space	227	343	E3	ä
186	272	BA	`	228	344	E4	ã
187	273	BB	§	229	345	E5	å
188	274	BC	°	230	346	E6	æ
189	275	BD	¨	231	347	E7	ç
190	276	BE	´	232	350	E8	é
191	277	BF	↑	233	351	E9	è
192	300	C0	Á	234	352	EA	ê
193	301	C1	À	235	353	EB	ë
194	302	C2	Â	236	354	EC	í
195	303	C3	Ä	237	355	ED	ì
196	304	C4	Ã	238	356	EE	î
197	305	C5	Å	239	357	EF	ï
198	306	C6	Æ	240	360	F0	ñ
199	307	C7	Ç	241	361	F1	ó
200	310	C8	É	242	362	F2	ò
201	311	C9	È	243	363	F3	ô
202	312	CA	Ê	244	364	F4	ö
203	313	CB	Ë	245	365	F5	õ
204	314	CC	Í	246	366	F6	ø
205	315	CD	Ì	247	367	F7	œ
206	316	CE	Î	248	370	F8	ú
207	317	CF	Ï	249	371	F9	ù
208	320	D0	Ñ	250	372	FA	û
209	321	D1	Ó	251	373	FB	ü
210	322	D2	Ò	252	374	FC	ß
211	323	D3	Ô	253	375	FD	space
				254	376	FE	space
				255	377	FF	space



Glossary

Access mode

The method of record retrieval and storage. The three Interactive COBOL access modes are SEQUENTIAL, RANDOM, and DYNAMIC. Access mode is specified in a SELECT clause in the Environment Division.

Actual decimal point

The physical representation of the decimal point position in a data-item, using either of the decimal point characters, period (.) or comma (,).

Alphanumeric edited item

Data-item restricted to combinations of the following symbols: A, X, 9, B, and slash (/). The contents can be any character in the COBOL character set except the underscore (_).

Alternate record key

A key, other than the primary record key, the contents of which identifies one or more records within an indexed file.

Assumed decimal point

An implied decimal point position that is indicated with the symbol V in the PICTURE description.

Block

A physical unit of data composed of one or more records. The COBOL concept of blocking (FD, BLOCK CONTAINS clause) does not apply to the Interactive COBOL system. Records are read and written without regard for the block boundaries.

Character, editing

Used to indicate a value to be inserted or replaced in a field. The 12 editing characters are:

Character	Meaning	Character	Meaning
B	Space	Z	Zero suppression with spaces
0	Zero	*	Zero suppression with *
+	Plus sign	$	Currency sign
–	Minus sign	,	Comma
CR	Credit symbol	.	Decimal point (period)
DB	Debit symbol	/	Slash

Character, picture

Any of the following characters, which can be used in a PICTURE string:

0	A zero insertion position
9	A decimal digit position
A	An alphabetic character position

B	A space insertion position
P	An assumed decimal scaling position
S	The presence of an operational sign
V	The location of an assumed decimal point
X	A position that may contain any character
CR DB + -	Editing sign controls
$	A leading currency sign position
/	A slash insertion position
,	A comma insertion position
.	A decimal point insertion position
*	A leading asterisk insertion position when the content is zero

Character, relation
Any of the three characters: less than (<), greater than (>), and equal to (=). They can be used in combination, for example, <=, meaning "less than or equal to."

Clause
Information in a COBOL program is written in clauses in the Identification, Environment, and Data divisions. A clause specifies an attribute of an *entry,* which is a series of clauses ending with a period.

Comment entry
An entry in the Identification Division that provides program documentation (the purpose of the program, its functions, etc.). It can consist of any number of lines of text made up of the characters in the data set.

Condition, class
The class condition ALPHABETIC or NUMERIC. In a program, an item can be tested to determine whether it is entirely alphabetic or numeric.

Condition, combined
Connecting two or more conditions to form one expression by using the logical operators AND or OR.

Condition, complex
Combining of conditions with the logical operators NOT, AND, and OR, and using optional parentheses to indicate the order of evaluation. Complex conditions occur in two forms: the negated simple condition and the combined condition.

Condition, relation
A condition comparing two operands. Each operand can be the data-item referenced by an identifier, a literal, or the value of an arithmetic expression. A relation condition has a value of true if the relation exists between the operands.

Condition, sign
The condition that determines if the algebraic value of a numeric item is less than, greater than, or equal to zero.

Condition, simple
A relation, class, switch-status, or sign condition.

Condition, switch-status

A condition that determines the on or off status of a switch defined in the Environment Division. The switch-name and the condition-name associated with the switch must be named in the SPECIAL-NAMES paragraph.

Conditional expression

Specifies a condition for testing within a program. Depending on the truth value of the condition, the program selects between alternative paths of processing. Conditional expressions are specified in IF and PERFORM statements. See *Condition, simple* and *Condition, complex*.

Conditional statement

Specifies that the truth value of a condition is to be determined and that the subsequent action of the program is dependent on this truth value.

Condition-name

(1) A condition-name is a programmer-defined word assigned to a program switch in the Environment Division. If the switch is ON, the associated condition-name is true and the OFF condition-name is false. (2) A word that is associated with a data-item by being assigned to one or more values that the data-item can assume. The condition-name is used instead of the relation condition.

Current record

The record that is available for manipulation in the area associated with a file.

Current record pointer

A conceptual item indicating a logical position in an indexed or relative file. It is used to select the next record to be read.

Dynamic access

An access mode in which specific logical records in an indexed or relative file can be read or written in a nonsequential manner, or read in a sequential manner. This must occur during the scope of a single OPEN statement.

Elementary item

A data-item described in the Data Division as having no logical subdivisions. See *Group item*.

Entry

Any set of consecutive clauses terminated by a period that appear in the Identification, Environment, or Data divisions.

Figurative constant

A reserved word that represents a specific constant value. Figurative constants are used in a program to avoid repetitive coding. Singular and plural forms are interchangeable; they must not be enclosed in quotation marks. The figurative constants and their values are:

ZERO, ZEROS, ZEROES	The character 0
SPACE, SPACES	The space character

HIGH-VALUE, HIGH-VALUES	Octal value 377
LOW-VALUE, LOW-VALUES	Octal value 000
QUOTE, QUOTES	The quotation mark character (")
ALL *literal*	The specified literal, which is a nonnumeric literal or a figurative constant. However, using a figurative constant with ALL is redundant.

Filename, external
The name by which a file is known to the operating system. An external filename is related to the internal filename in a SELECT clause in the Environment Division.

Filename, internal
The name by which a file is know to a COBOL program. An internal filename is assigned to an external file by using the SELECT clause. The program refers to the internal name whenever I/O statements are processed. Filename can be up to 30 characters in length and can consist of uppercase letters and the hyphen. Within a program, filenames must be unique.

File, relative
A file with an ORGANIZATION IS RELATIVE clause in its FD. The records in this type of file have keys in ascending sequence relative to the first record in the file. A relative file can be accessed sequentially, randomly, or dynamically.

File, sequential
A file with an ORGANIZATION IS SEQUENTIAL clause in its FD. The records in this type of file are accessed in physical sequence. Keys cannot be used.

Fixed-length record
A record in a fixed-length sequential file. Each record in the file is the same length. This length is determined at the time the file is created and cannot be changed.

Group item
A named contiguous set of elementary or group items.

id
An abbreviation for the common COBOL term *identifier*.

Identifier
A data-name with any necessary qualifiers, subscripts, or indices to make a unique reference to a data-item.

id-lit
An abbreviation for the common COBOL construct *identifier-literal* and indicates that either an identifier or a literal can be used.

Imperative statement
A statement that begins with a verb that specifies an unconditional action to be taken by the program.

In-line PERFORM statement

A PERFORM statement that does not include a *procedure–name* variable. Interactive COBOL executes only the statements contained within the PERFORM statement.

Index

A data-item or a simple expression that points to an element of a table. If an index is an expression, it must consist of either a data-item plus an integer or a data-item minus an integer.

Indexed file

A file in which records are identified by a key value rather than by physical or logical position. At the operating system level, indexed files comprise a data portion and an index portion. The data portion contains the file's records. The index portion contains key values and pointers to records for one primary key and, optionally, up to four alternate keys.

ISAM

Indexed Sequential Access Method. The term *ISAM file* commonly refers either to an indexed or relative file.

Key of reference

The primary or alternate key currently being used to access records within an indexed file.

Line numbers

In the Screen Section, line numbers specify the lines on the display screen. The LINE clause in the Screen Section allows you to specify on which line number of the display screen an item is to appear.

Line terminator

The line terminator for AOS, AOS/VS, UNIX systems, and DG/RDOS is New Line; for RDOS it is carriage return (CR).

Logical operator

One of the reserved words AND, OR, or NOT. In forming conditionals, AND or OR can be used as connectives and NOT is used for logical negation.

Next record

The record that logically follows the current record in a file. For sequential files this is the next record according to the order in which the records were originally written. For other files the next record is determined by the key used.

Numeric edited item

A data-item restricted to combinations of the symbols B, slash (/), P, V, Z, 0, 9, comma, period, asterisk, minus, CR, DB, and CS (currency symbol). The item can be up to 18 characters long. The PICTURE must contain at least one 9 or Z and one of the edit symbols.

Operational sign

An algebraic sign associated with an item to indicate whether its value is positive or negative.

Out-of-line PERFORM statement
A PERFORM statement that directs program control to *procedure-name*. Interactive COBOL executes each set of instructions contained in each *procedure-name* referred to by the PERFORM statement.

Paragraph
In the Procedure Division, a paragraph-name followed by one or more sentences. In the other divisions, a paragraph header followed by one or more entries.

Paragraph-name
A programmer-defined name that identifies one or more sentences in the Procedure Division. Paragraph-name can contain the hyphen, up to 30 characters from the set of uppercase letters, and the digits 0-9.

Phrase
Two or more words that are considered a syntactical unit. It is part of a clause in the Identification, Environment, and Data divisions and part of a statement in the Procedure Division.

Primary key
A key whose contents uniquely identifies a record within an indexed file.

Relational operator
Reserved word or relational character used to construct conditional statements.

Relative key
A key whose contents identify the position of a record in a relative file. It must have a PICTURE of 9(4) to 9(18) COMP.

Run unit
A logical set of programs that includes the main program and all sub-programs that have been called but not cancelled.

Screen-name
A data-name that identifies an item in the Screen Section of the Data Division.

Section
Within a division of a COBOL program, a logical unit combining entries and sentences. In the Environment and Data divisions, a section consists of a section header and zero or more entries. The section header contains a reserved section name followed by the word SECTION. A section header must be followed by a period and a space. In the Procedure Division, a section consists of a programmer-defined section name, followed by a period and space, and zero or more paragraphs.

Sentence
A sequence of one or more statements, the last of which is terminated by a period and a space.

Separator
One of the following characters used to terminate a character string: Space, tab, comma, semicolon, period, left and right parentheses, and quotation marks.

Statement

In the Procedure Division, one or more phrases that specify an action to be taken by the COBOL program.

Variable origin screen

The ACCEPT and DISPLAY statements can contain LINE and COLUMN clauses that specify variable origin. When these clauses are included, the entire screen is offset by the values specified. For example, DISPLAY SCR-1 LINE 10 COLUMN 5 offsets all fields 10 lines down and 5 columns across. Normally, variable origin screens are zero-based. Thus, if SCR-1 starts at line 1 column 1, DISPLAY SCR-1 AT LINE 1 COLUMN 1 displays the screen-name at line 2 column 2.

Variable-length record

A record in a variable sequential file. Each record in the file contains a 2-byte header that specifies the length of the record. The length of each record must fall between the minimum and maximum established in the RECORD CONTAINS clause of the FD entry for the file.

End of Glossary

Index

Symbols

.NX (.nx) portion, 2-3, 2-4, 2-7

.XD (.xd) portion, 2-3, 2-4, 2-7

Numbers

0 symbol, in PICTURE clause, 1-7, 7-15

01 level, 7-12

02-49 levels, 7-12

77 level, 1-4, 4-1, 4-3, 7-12

88 level, 1-4, 1-25, 4-1, 4-3, 7-13

9 symbol, in PICTURE clause, 1-5, 1-7, 7-15

A

A symbol, in PICTURE clause, 1-5, 1-7, 7-15

ACCEPT, 1-20, 2-3, 8-3
  identifier, 8-5
  in Screen Section, 3-1, 3-6, 3-14, 3-16, 7-5
  screen, 8-4, Glossary-7
  system, 8-5
  termination in, 3-18, 8-4, 8-6
  timeout, 8-4, 8-57

Access mode, Glossary-1
  dynamic, 1-30, 2-5, Glossary-3
  random, 2-4
  sequential, 2-4

ACCESS MODE clause, 2-5, 6-5

ADD, 1-20, 8-9
  substituting for COMPUTE, 8-20

ADD CORRESPONDING, 1-20, 1-23, 8-10

Alignment rules, 1-11

alphabet-name IS clause, 6-2

Alphabetic item
  alignment of, 1-11
  described in PICTURE clause, 1-5

Alphanumeric edited item, 1-4, Glossary-1

Alphanumeric item
  alignment of, 1-11
  described in PICTURE clause, 1-7

Alternate key, 2-6
  specification of, 6-5

Alternate record key, Glossary-1

ALTERNATE RECORD KEY clause, 2-3, 2-6, 6-5

ANALYZE utility, 2-5, 2-6, 2-7

Area A
  in card format, 4-3
  in CRT format, 4-1

Area B
  in card format, 4-3
  in CRT format, 4-1

Arithmetic expressions, 1-20
  evaluation order in, 1-21

Arithmetic operations, verbs used in, 1-20

Arithmetic operators, 1-21
  table of, 1-21

ASCII character set, 1-2
  8-bit, 1-2

ASSIGN clause, 6-3

Asterisk
  as comment indicator, 4-4
  in PICTURE clause, 1-10, 7-15

AT END, and USE procedure, 8-85

AT END condition, 1-31

AUTHOR paragraph, 5-1

AUTO clause, 3-15

B

B symbol, in PICTURE clause, 1-5, 1-7, 7-15

BACKGROUND-COLOR clause, 3-15
  order of execution, 3-16, 7-9

BELL clause, 3-15, 7-10
  order of execution, 3-16

BLANK LINE clause, in Screen Section, 3-14, 7-10
  order of execution, 3-16

Blank lines in source code, 4-4

BLANK SCREEN clause, 3-14
  order of execution, 3-16

BLANK WHEN ZERO clause, 3-16, 7-10, 7-20
  illegal with USAGE IS INDEX, 7-17
  no effect on SECURE, 3-15
  not used with *, 1-11

BLINK clause, 3-16, 7-10
  order of execution, 3-16, 7-9

Block, Glossary-1

BLOCK CONTAINS clause, 7-2

BACKGROUND clause, 7-10

BRIGHT clause, 7-10
  in Screen Section, 3-16

C

CALL, 1-20, 8-12
  switches in calling program, 8-13

CALL PROGRAM, 1-20, 8-14
  EXCEPTION STATUS item in, 8-6
  with USING phrase, 7-4

Called program, 7-3
  example of, 7-4, 8-12, 8-15
  level numbers of receiving items in, 7-4

Calling program, 7-3
  example of, 7-4, 8-12, 8-15
  level numbers of sending items in, 7-4
  using switches, 1-27

CANCEL, 1-20, 8-17

Card format, 4-2

Character
  editing, Glossary-1
  picture, 1-4, Glossary-1
  relation, Glossary-2

Character set, 1-2
  8-bit, 1-2
  ASCII, 1-2
  COBOL, 1-2

Class condition, 1-26, Glossary-2
  with incompatible data, 1-26

Class test
  ALPHABETIC, 1-26
  NUMERIC, 1-26

Clause, 1-1, Glossary-2
  in Screen Section, 3-9

CLOSE, 1-20, 8-18
  organization of files with, 8-18
  unsuccessful, 1-30, 8-18
  with DELETE FILE, 8-18
  with LOCK, 8-18

COBOL
  character set, 1-2
  clause, 1-1, Glossary-2
  divisions of, 1-1
  entry, 1-1, Glossary-3
  paragraphs, 1-1, 1-17, Glossary-6
  sections, 1-1, 1-17, Glossary-6
  sentences, 1-1, 1-18, Glossary-6
  separators, 1-2
  source code, creating, 4-1
  statements, 1-1, 1-18, Glossary-7
    conditional, 1-19
    imperative, 1-19
  verbs, table of, 1-20
  words, 1-2

CODE-SET clause, 7-2

COLLAPSE utility, 2-5

COLUMN clause, 3-11, 7-10
  order of execution, 3-16
  with DISPLAY, 8-25

Column positioning
  absolute, 3-12, 3-13
  relative, 3-12
  sequence rules, 3-14
  table, 3-13

Combined condition, 1-28, 8-33, Glossary-2

Comma, in PICTURE clause, 1-7, 7-15

Comment area, in card format, 4-3

Comment entry, Glossary-2

Comment line, 4-4
  in card format, 4-3
  in CRT format, 4-1

Comments, in Identification Division, 5-1

Compile-time switch, 2-7

Compiler-directing, verbs, 1-20

Complex condition, 1-27, Glossary-2

COMPUTE, 1-20, 8-19
  intermediate results with, 8-19
  maximum size of operands with, 8-19
  used for exponentiation, 8-19

Condition, 1–23, 8–33
  abbreviated combined relation, 1–28, 1–29
  class, 1–26, Glossary–2
  combined, 1–28, 8–33, Glossary–2
  complex, 1–27, Glossary–2
  condition–name, 1–4, 1–25, Glossary–3
  evaluation order of, 1–29
  exception, 1–30
  negated combined, 1–28
  negated simple, 1–28
  relation, 1–24, Glossary–2
  sign, 1–27, Glossary–2
  simple, 1–23, Glossary–2
  switch–status, 1–27, Glossary–3

Condition–name condition, 1–25, Glossary–3

Conditional expression, 1–23, Glossary–3

Conditional statement, 1–19, Glossary–3

Configuration Section, 6–2

Constants, figurative, 1–3, Glossary–3

Continuation lines, 4–3

COPY, 1–20, 8–21
  location in source program, 8–21
  with COPY files, 8–21
  with indexed files, 8–21

COPY ... IN DICTIONARY, 1–20

CORRESPONDING phrase, 1–23

CR symbol, in PICTURE clause, 1–8, 7–15

CRT format, 4–1
  with COPY files, 8–21

CURRENCY SIGN clause, 1–8, 6–2

Currency symbol, in PICTURE clause, 1–8, 1–9, 7–15

Current record, Glossary–3

Current record pointer, 1–30, Glossary–3
  and DELETE statement, 8–23
  and OPEN statement, 8–45
  and READ statement, 8–56, 8–57, 8–58, 8–59
  and REWRITE statement, 8–61
  and START statement, 8–68, 8–69, 8–70
  and UNDELETE statement, 8–79
  and UNLOCK statement, 8–80
  and WRITE statement, 8–87

D

Data description entry, 7–12
  in Screen Section, 3–1

Data Division, 1–1, 7–1
  duplicate names in, 1–12

Data–item
  alignment of, 1–11
  elementary, 1–4, Glossary–3
  in Screen Section, 3–4, 7–7
  length of, 7–18

Data–name
  clause in Data Division, 7–13
  qualification of, 1–12, 1–13

DATA RECORD clause, 7–2

DATA SIZE clause, 6–6

DATE, in system ACCEPT, 8–5

DATE–COMPILED paragraph, 5–1

DATE–WRITTEN paragraph, 5–1

DAY, in system ACCEPT, 8–5

DB symbol, in PICTURE clause, 1–8, 7–15

Decimal point
  actual, Glossary–1
  assumed, Glossary–1
  in PICTURE clause, 1–7

DECIMAL–POINT IS COMMA clause, 1–8, 6–2, 7–15

Declaratives section, 1–17, 8–1, 8–84
  and unsuccessful OPEN or CLOSE, 1–30
  I/O processing, 1–30

DELETE, 1–20, 2–5, 8–23
  and current record pointer, 8–23
  INVALID KEY error in, 1–31
  used with records, 8–23

DELETE FILE statement, 8–24
  with CLOSE, 8–18

Delimiter, in COBOL program, 1–2

DIM clause, 7–10
  in Screen Section, 3–16

DISK, in ASSIGN clause, 2–2, 6–3

DISPLAY, 1–20, 2–3, 8–25
  in ASSIGN clause, 2–2, 6–3
  in Screen Section, 3–1, 3–4, 3–10, 3–14, 3–16, 7–5, Glossary–7
  with P symbol, 8–26

DISPLAY items, 7–16

Displaying data, with P symbol, 1–6

DIVIDE, 1-20, 8-27
substituting for COMPUTE, 8-20

Divisions of COBOL program, 1-1

Dollar sign, in PICTURE clause, 1-8, 1-9

Duplicate names, in Data and Procedure divisions, 1-12

DUPLICATES phrase, 6-5

Dynamic access, 2-5, Glossary-3

E

Editing
fixed insertion, 1-8
floating insertion, 1-9
in PICTURE clause, 1-7, Glossary-1
simple insertion, 1-7
special insertion, 1-7
zero supression, 1-10

Eight-bit ASCII character set, 1-2

Elementary item, 1-4, Glossary-3

END-, 8-2

Entering table data
example of interactive loading, 1-16
from a data file, 1-15
interactively, 1-15

Entry, 1-1, Glossary-3

ENVIRONMENT, in system ACCEPT, 8-6

ENVIRONMENT data
CONSOLE-TYPE, 8-6
PID, 8-6
PROGRAM-NAME, 8-6
REVISION-CODE, 8-6
SYSTEM-CODE, 8-6

Environment Division, 1-1, 6-1

ERASE EOL clause, 3-14, 7-10
order of execution, 3-16

ERASE EOS clause, in Screen Section, 3-14, 7-10

ERASE LINE clause, in Screen Section, 3-14, 7-10
order of execution, 3-16

ESCAPE KEY, in system ACCEPT, 8-5

Exception condition, I/O, 1-30

EXCEPTION STATUS, in system ACCEPT, 8-6

EXCLUSIVE phrase, 8-46

EXIT, 1-20, 8-29

EXIT PROGRAM, 1-20, 8-13, 8-30

Explicit scope delimiters, 8-2

Exponentiation, and COMPUTE, 8-19

EXTEND phrase, 8-46

External filename, 5-1, Glossary-4

F

FD entry, 7-2
multiple 01 levels in, 7-14

Figurative constant, 1-3, Glossary-3

File
access mode and assignment of, 6-4
dynamic access, 2-5
in COBOL program, 1-4
indexed, 2-3, 6-4
organization, table of, 2-1
organization and access, 2-1
random access, 2-4
relative, 2-7, 6-4, Glossary-4
sequential, 2-1, 6-4, Glossary-4
sequential access, 2-4

File identifiers, open file limits, 8-47

File processing, termination of, 8-18

File Section, 7-2, 7-21

FILE STATUS clause, 1-30, 6-6

File Status code, 1-30

File terminators, line sequential files, 2-2

FILE-CONTROL paragraph, 6-3

FILECALC utility, 2-1, 2-5, 2-7

Filename
external, 5-1, 6-5, Glossary-4
internal, Glossary-4

FILESTATS utility, 2-1, 2-5

FILLER clause, 7-12, 7-13
illegal in Screen Section, 3-8

Fixed insertion editing, 1-8

Fixed sequential file, 2-1
access mode and assignment of, 6-4

Fixed-length record, 2-1, Glossary-4

Floating insertion editing, 1-9

FOREGROUND-COLOR clause, order of execution, 3-16, 7-9

FROM clause, 7-10
  in Screen Section, 3-11

FULL clause, 3-15

FOREGROUND clause, 7-10

G

GO TO, 1-20, 8-31

Group, entry in Screen Section, 7-8

Group item, 1-4, Glossary-4
  and VALUE clause, 7-22

H

Hierarchy, 1-4

HIGHLIGHT clause, 7-10
  in Screen Section, 3-16
  order of execution, 3-16, 7-9

HIGH-VALUES, 1-3
  illegal value for key, 8-23, 8-56, 8-62, 8-69, 8-87

Hyphen, as continuation indicator, 4-3

I

I-O-CONTROL paragraph, 6-6

I/O exception condition, 1-30

I/O operations
  Declaratives section, 1-30
  EXCEPTION STATUS, 8-6
  FILE STATUS, 6-6
  verbs used in, 1-20

ICOMPACT utility, 2-5

id, Glossary-4

id-lit, Glossary-4

Identification Division, 1-1, 5-1

Identifier, Glossary-4
  see also Qualifiers, 1-12

IF, 1-20, 8-33

IMMEDIATE option
  with REWRITE, 8-61
  with WRITE, 8-86

Imperative statement, 1-19, Glossary-4

Implicit scope delimiters, 8-2

Index, Glossary-5
  in table, 1-13

INDEX items, 7-17

INDEX SIZE clause, 6-6

INDEXED BY phrase, 1-14, 7-19

Indexed file, 2-3, Glossary-5
  access mode and assignment of, 6-4
  alternate key, 2-5
  assignment to device, 6-4
  data portion, 2-4
  index portion, 2-4
  primary key, 2-5
  SELECT clause with, 6-4
  updating, 2-6

Indicator area
  in card format, 4-2
  in CRT format, 4-1

Indicator character, 4-1, 4-2

Input-Output Section, 6-3

INSPECT, 1-20, 8-36
  overlapping operands in, 1-22

INSTALLATION paragraph, 5-1

Intermediate results, in COMPUTE statement, 8-19

Internal filename, Glossary-4

INVALID KEY condition, 1-31

INVALID KEY phrase
  and DELETE statement, 8-23
  and primary key duplicates, 2-6
  and READ statement, 8-59
  and REWRITE statement, 8-62
  and START statement, 8-69
  and UNDELETE statement, 8-79
  and USE procedure, 8-58, 8-85
  and USE statement, 1-31
  and WRITE statement, 8-88

ISAM, Glossary-5

J

JUSTIFIED clause, 7-10, 7-20
  alignment of data with, 1-11
  illegal with USAGE IS INDEX, 7-17
  in Screen Section, 3-16
  no effect on SECURE, 3-15

K

Key
  alternate, 2-6

duplicate, 2-6
HIGH-VALUES illegal in, 8-23, 8-56, 8-62, 8-69, 8-87
length of, 2-1, 2-6
primary, 2-6, Glossary-6
relative, 2-7, 6-6, Glossary-6

Key of reference, 8-59, Glossary-5

KEYBOARD, in ASSIGN clause, 2-2, 6-3

L

LABEL clause, 7-2

Level number, 1-4, 7-12
correct area with, 4-1, 4-3
in Screen Section, 3-1, 7-8
level 01, 7-12
level 77, 1-4, 7-12
level 88, 1-4, 1-25, 7-13
levels 02-49, 7-12
of items in called and calling programs, 7-4

LINE clause, 3-11
order of execution, 3-16
with DISPLAY, 8-25

LINE NUMBER, in system ACCEPT, 8-5

Line number, on display screen, Glossary-5

Line positioning
absolute, 3-12, 3-13
PLUS phrase, 3-12
relative, 3-12
sequence rules, 3-14
table, 3-13

Line sequential file, 2-1, 6-5
access mode and assignment of, 6-4
assignment to device, 6-4
creating on disk, 6-5

Line terminator, Glossary-5

Linkage Section, 7-3, 7-21
size of, 7-4

Literal
entry in Screen Section, 3-3
nonnumeric, 1-3
numeric, 1-3

LOCK option
in CLOSE statement, 8-18
in READ statement, 8-56

Logical operators, 1-28, Glossary-5

LOWLIGHT clause, 7-10
in Screen Section, 3-16
order of execution, 3-16, 7-9

LOW-VALUES, 1-3

M

Maximum size, of operands, 8-19

MEMORY SIZE clause, 6-2

Minus sign
as continuation indicator, 4-3
in PICTURE clause, 1-8, 1-9, 7-15

MOVE, 1-20, 8-39
overlapping operands in, 1-22

MOVE CORRESPONDING, 1-20, 1-23, 8-41

Moving data, with P symbol, 1-6

MULTIPLY, 1-20, 8-43
substituting for COMPUTE, 8-20

N

Name, programmer-defined, 1-1

Negated combined condition, 1-28

Negated simple condition, 1-28

Next record, Glossary-5

Nondeclarative format, 8-1

Nonnumeric literal, 1-3

Numeric edited item, Glossary-5

Numeric item
alignment of, 1-11
described in PICTURE clause, 1-5
maximum length of, 1-5

Numeric literal, 1-3

O

OBJECT-COMPUTER paragraph, 6-2

OCCURS clause, 1-13, 7-13, 7-19
examples of use, 7-19
illegal with VALUE clause, 7-19, 7-21
legal level numbers with, 7-19
nested, 7-19

OPEN, 1-20, 8-45
and current record pointer, 8-45
EXTEND option, 2-3

I-O option, 2-3
unsuccessful, 1-30

Open file limits, 8-47

Operand
maximum size of, 8-19
overlapping, 1-22

Operational sign, Glossary-5, Glossary-6

Operators
arithmetic, 1-21
logical, 1-28, Glossary-5

ORGANIZATION clause, 2-2, 6-5

Overpunched signs, 7-18

P

P symbol
and DISPLAY statement, 8-26
displaying data with, 1-6
in PICTURE clause, 1-6, 7-15
moving data with, 1-6
position of, 1-6
size of data-item with, 1-6
with VALUE clause, 1-6

Paragraph, 1-1, 1-17, Glossary-6

Paragraph-name, Glossary-6
qualification of, 1-13

PERFORM, 1-14, 1-20, 8-48

PERFORM statement
in-line, 8-49
out-of-line, 8-49

PERFORM UNTIL, 1-23

Period, in PICTURE clause, 1-7, 7-15

Phrase, 1-1, Glossary-6

PICTURE clause, 1-4, 7-14
actual decimal point, Glossary-1
alphabetic data-items, 1-5
alphanumeric data-items, 1-7
alphanumeric edited item, Glossary-1
assumed decimal point, Glossary-1
characters used in, Glossary-1
editing in, 1-7, Glossary-1
illegal with USAGE IS INDEX, 1-4, 7-14, 7-17
in Screen Section, 3-4, 3-10
numeric data-items, 1-5
symbols used in, 7-15

Plus sign, in PICTURE clause, 1-8, 1-9, 7-15

Primary key, 2-6, Glossary-6

PRINTER, in ASSIGN clause, 2-2, 6-3

PRINTER-1, in ASSIGN clause, 6-3

Procedure Division, 1-1, 1-17, 8-1
duplicate names in, 1-12
terminating statements, 8-2

Procedure Division Using header, 7-4, 8-13

PROGRAM COLLATING SEQUENCE clause, 6-2

Program control
CALL, 8-12
CALL PROGRAM, 8-14
Declaratives section, 1-17, 8-1
transfer of control, 1-32
verbs used in, 1-20
with AT END, 1-31
with INVALID KEY, 1-31

PROGRAM-ID paragraph, 5-1

programmer-defined name, 1-1

programmer-defined words, 1-2

Programs, creating, 4-1

Q

Qualification of names, 1-12
in Data Division, 1-13
in Procedure Division, 1-13

Qualifiers, 1-12
format of, 1-12
separation of, 1-12

QUOTE (figurative constant), 1-3

R

Random access, 2-4

READ, 1-20, 2-1, 2-6, 8-55
AT END condition with, 1-31
and current record pointer, 8-56, 8-57, 8-58, 8-59
INVALID KEY error in, 1-31
line sequential file, 2-3

Record
01 level, 1-4
deletion of, 2-5
fixed-length, 2-1, Glossary-4
in COBOL program, 1-4
length of, 2-1
variable-length, 2-1, Glossary-7

RECORD CONTAINS clause, 2-2, 7-2

Record key, alternate, Glossary-1

RECORD KEY clause, 2-3, 6-5

RECORDING MODE clause, 2-2, 7-2

REDEFINES clause, 7-12, 7-13, 7-14, 7-19
  illegal with VALUE clause, 7-21
  implied by multiple FD 01-levels, 7-14

Relation condition, 1-24, Glossary-2
  characters used in, Glossary-2

Relational operator, 1-29, Glossary-6

Relative file, 2-7, Glossary-4
  access mode and assignment of, 6-4
  assignment to device, 6-4
  key in, 2-7, Glossary-6
  SELECT clause with, 6-4

Relative key, Glossary-6

REORG utility, 2-5
  changing file organization, 6-5

REQUIRED clause, 3-15, 7-11

Reserved words, 1-2
  END- verbs, 8-2

REWRITE, 1-20, 2-6, 8-61
  and current record pointer, 8-61
  INVALID KEY error in, 1-31

ROUNDED phrase, 1-21, 1-22
  with DIVIDE statement, 8-28

Run unit, Glossary-6

REVERSE-VIDEO clause, 7-11
  in Screen Section, 3-16
  order of execution, 3-16, 7-9

S

S symbol, in PICTURE clause, 1-5, 7-15

SAME [RECORD] AREA clause, 6-6
  and REWRITE, 8-61

Scaling character P, and DISPLAY statement, 8-26

Screen Section, 3-1, 7-5
  ACCEPT statement, 3-1, 3-6, 3-14, 8-4
  AUTO clause in, 3-15
  BACKGROUND clause in, 7-10
  BACKGROUND-COLOR clause in, 3-15
  BELL clause in, 3-15
  BLANK LINE clause in, 3-14, 7-10
  BLANK SCREEN clause in, 3-14
  BLANK WHEN ZERO clause in, 3-16, 7-10
  BLINK clause in, 3-16
  BRIGHT clause in, 3-16, 7-10
  clauses in, 3-9
  COLUMN clause in, 3-11, 7-10
  data description entry in, 3-1
  data-item entry in, 3-4, 7-7
  DIM clause in, 3-16, 7-10
  DISPLAY clause in, 3-10
  DISPLAY statement, 3-1, 3-4, 3-10, 3-14
  ERASE EOL clause in, 3-14, 7-10
  ERASE EOS clause in, 3-14, 7-10
  ERASE LINE clause in, 3-14, 7-10
  example of, 3-18
  FILLER illegal in, 3-8
  FOREGROUND clause in, 7-10
  formats for entries, 7-5
  FROM clause in, 3-11
  FULL clause in, 3-15
  group entry in, 3-7, 7-8
  HIGHLIGHT clause in, 3-16, 7-10
  JUSTIFIED clause in, 3-16
  level number in, 3-1, 7-8
  LINE clause in, 3-11
  literal entry in, 3-3
  LOWLIGHT clause in, 3-16, 7-10
  order of clause execution, 3-16
  PICTURE clause in, 3-4, 3-10
  placement in a COBOL program, 3-1
  REQUIRED clause in, 3-15, 7-11
  REVERSE-VIDEO clause in, 3-16
  REVERSE-VIDEO clause in, 7-11
  SECURE clause in, 3-15
  SIGN clause in, 7-11
  storage in, 3-17
  TO clause in, 3-11
  UNDERLINE clause in, 3-16
  UNDERLINE clause in, 7-11
  USAGE clause in, 7-11
  USING clause in, 3-11, 7-11
  VALUE clause in, 3-10
  variable origin screen, Glossary-7

Screen-name, Glossary-6

SEARCH, 8-63

Section, 1-1, 1-17, Glossary-6
  used in qualification, 1-13

SECURE clause, 3-15
  BLANK WHEN ZERO no effect on, 3-15
  JUSTIFIED no effect on, 3-15

SECURITY paragraph, 5-1

Segment-number, 1-18

SELECT clause, 6-3
  with external filename, 6-5

with indexed files, 2–3, 6–4
with relative files, 6–4
with sequential files, 6–3

Sentence, 1–1, 1–18, Glossary–6

Separator, 1–2, Glossary–6

Sequence number area, in card format, 4–2

Sequential access, 2–4

Sequential file, 2–1, Glossary–4
access mode and assignment of, 6–4
assignment to device, 6–4
creating on disk, 6–5
fixed–length, 2–2
line, 2–2
SELECT clause with, 6–3
variable–length, 2–2

SET, 1–14, 1–20, 8–67
overlapping operands in, 1–22

SIGN clause, 1–26, 7–11, 7–17

Sign condition, Glossary–2

Simple condition, 1–23, Glossary–2

Simple insertion editing, 1–7

SIZE ERROR
with ADD CORRESPONDING statement, 8–10
with ADD statement, 8–9
with DIVIDE statement, 8–28
with MULTIPLY statement, 8–44
with SUBTRACT CORRESPONDING statement, 8–77

SIZE ERROR phrase, 1–21, 1–23

Slash (/)
as comment indicator, 4–4
as form feed, 4–4
in PICTURE clause, 1–7, 7–15

Source code
blank lines in, 4–4
card format, 4–2
comment lines in, 4–4
continuation lines in, 4–3
creating, 4–1
CRT format, 4–1

SOURCE–COMPUTER paragraph, 6–2

SPACE (figurative constant), 1–3

Special insertion editing, 1–7

SPECIAL–NAMES paragraph, 1–8, 6–2
with switch–status condition, 1–27

STRING, 8–72

START, 1–20, 2–6, 8–68
and current record pointer, 8–68, 8–69, 8–70
INVALID KEY error in, 1–31

Statement, 1–1, 1–18, Glossary–7
conditional, 1–19
imperative, 1–19

STOP, 1–20, 8–71

Subscript, 1–13

SUBTRACT, 1–20, 8–75
substituting for COMPUTE, 8–20

SUBTRACT CORRESPONDING, 1–20, 1–23, 8–77

Switch
compile–time, 2–7
in calling program, 8–13
logical, 1–27, 6–2

SWITCH literal IS clause, 6–2

Switch–status condition, 1–27, Glossary–3

SYNCHRONIZED clause, 7–20
illegal with USAGE IS INDEX, 7–17

System calls, 8–15
invalid with USING, 8–15

T

Table, 1–13, 7–19
manipulation using PERFORM VARYING, 1–17
multidimensional, 7–19
referencing, 1–16

Terminating statements, 8–2

Termination
of file processing, 8–18
program, 1–30

Termination codes, in ACCEPT, 8–6

TIME, in system ACCEPT, 8–5

TIMEOUT, in ACCEPT, 8–4

TIMEOUT, in READ, 8–57

TO clause, 7–11
in Screen Section, 3–11

U

UNDELETE, 1–20, 2–5, 8–23, 8–79
and current record pointer, 8–79
INVALID KEY error in, 1–31

UNDERLINE clause, 7-11
  in Screen Section, 3-16
  order of execution, 3-16, 7-9
Uniqueness of reference, 1-12
UNLOCK, 1-20, 8-80
  and current record pointer, 8-80
UNSTRING, 8-81
USAGE clause
  and ADD CORRESPONDING, 8-10
  and SUBTRACT CORRESPONDING, 8-77
  PIC illegal with INDEX items, 1-4
USE, 1-20, 8-84
  and DELETE, 8-23
  with OPEN and CLOSE, 1-30
USING clause, 7-11
  in Screen Section, 3-11
USING phrase, with CALL, 8-13
USAGE clause, 7-11, 7-16
  BINARY, 7-16
  COMPUTATIONAL, 7-16
  INDEX, 7-17
  PACKED-DECIMAL, 7-17
  PIC illegal with INDEX items, 7-14

V

V symbol, in PICTURE clause, 1-5, 7-15
VALUE clause, 1-14, 1-25, 7-21
  and data-item categories, 7-21
  at group level, 7-22
  illegal in File and Linkage sections, 7-21
  illegal with OCCURS clause, 7-19, 7-21
  illegal with REDEFINES clause, 7-21
  illegal with USAGE IS INDEX, 7-17
  in Screen Section, 3-10, 7-11
  with P symbol, 1-6
Variable sequential file, 2-1
  access mode and assignment of, 6-4
Variable-length record, 2-1, Glossary-7
Verbs
  END-, 8-2
  table of, 1-20

W

WITH DEBUGGING MODE clause, 6-2
Words
  in COBOL program, 1-2
  programmer-defined, 1-2
  reserved, 1-2
Working-Storage Section, 7-3
WRITE, 1-20, 2-6, 8-86
  and current record pointer, 8-87
  INVALID KEY error in, 1-31

X

X symbol, in PICTURE clause, 1-7, 7-15

Z

Z symbol, in PICTURE clause, 1-10, 7-15
Zero, in PICTURE clause, 1-7, 7-15
ZERO (figurative constant), 1-3
Zero suppression editing, 1-10
  and BLANK WHEN ZERO clause, 1-11

Related Documents

UNIX

Using Interactive COBOL on DG/UX™ and Interactive UNIX Systems	069–000359
Interactive COBOL Utilities (DG/UX™ and 386/ix™ Systems)	069–000358

AOS/VS and AOS/VS II

Using Interactive COBOL on AOS/VS	069–000224
Interactive COBOL Utilities (AOS/VS)	069–000308

AOS

Interactive COBOL User's Guide (AOS)	069–705015
Interactive COBOL Utilities (AOS)	069–705021

RDOS and DG/RDOS

Interactive COBOL User's Guide (RDOS, DG/RDOS)	069–705014
Interactive COBOL Utilities (RDOS, DG/RDOS)	069–705020

MS–DOS

Using Interactive COBOL on MS-DOS®	069–000252
Interactive COBOL Utilities (MS-DOS®)	069–000074

 Licensed Material – Property of Data General Corporation

TIPS ORDER FORM

Mail To: Data General Corporation
Attn: Educational Services/TIPS G155
4400 Computer Drive
Westboro, MA 01581 - 9973

BILL TO:	SHIP TO: (No P.O. Boxes - Complete Only If Different Address)
COMPANY NAME ______	COMPANY NAME ______
ATTN: ______	ATTN: ______
ADDRESS ______	ADDRESS (NO PO BOXES) ______
CITY ______	CITY ______
STATE ______ ZIP ______	STATE ______ ZIP ______

Priority Code ______ (See label on back of catalog)

Authorized Signature of Buyer ______ Title ______ Date ______ Phone (Area Code) ______ Ext. ______
(Agrees to terms & conditions on reverse side)

ORDER #	QTY	DESCRIPTION	UNIT PRICE	TOTAL PRICE

A SHIPPING & HANDLING

☐ UPS	ADD
1-4 Items	$ 5.00
5-10 Items	$ 8.00
11-40 Items	$ 10.00
41-200 Items	$ 30.00
200+ Items	$100.00

Check for faster delivery

Additional charge to be determined at time of shipment and added to your bill.

☐ UPS Blue Label (2 day shipping)
☐ Red Label (overnight shipping)

B VOLUME DISCOUNTS

Order Amount	Save
$0 - $149.99	0%
$150 - $499.99	10%
Over $500.00	20%

Tax Exempt #
or Sales Tax
(If applicable) ______

ORDER TOTAL	
Less Discount See B	-
SUB TOTAL	
Your local* sales tax	+
Shipping and handling - See A	+
TOTAL - See C	

C PAYMENT METHOD

☐ Purchase Order Attached ($50 minimum)
P.O. number is________. (Include hardcopy P.O.)
☐ Check or Money Order Enclosed
☐ Visa ☐ MasterCard ($20 minimum on credit cards)

Account Number ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐ Expiration Date ☐☐☐☐

______
Authorized Signature
(Credit card orders without signature and expiration date cannot be processed.)

THANK YOU FOR YOUR ORDER

PRICES SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.
PLEASE ALLOW 2 WEEKS FOR DELIVERY.
NO REFUNDS NO RETURNS.

* Data General is required by law to collect applicable sales or use tax on all purchases shipped to states where DG maintains a place of business, which covers all 50 states. Please include your local taxes when determining the total value of your order. If you are uncertain about the correct tax amount, please call 508-870-1600.

Form 702
Rev. 8/87

DATA GENERAL CORPORATION TECHNICAL INFORMATION AND PUBLICATIONS SERVICE TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form. These terms and conditions apply to all orders, telephone, telex, or mail. By accepting these products the Customer accepts and agrees to be bound by these terms and conditions.

1. CUSTOMER CERTIFICATION
Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub-licensee of the software which is the subject matter of the publication(s) ordered hereunder.

2. TAXES
Customer shall be responsible for all taxes, including taxes paid or payable by DGC for products or services supplied under this Agreement, exclusive of taxes based on DGC's net income, unless Customer provides written proof of exemption.

3. DATA AND PROPRIETARY RIGHTS
Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

4. LIMITED MEDIA WARRANTY
DGC warrants the CLI Macros media, provided by DGC to the Customer under this Agreement, against physical defects for a period of ninety (90) days from the date of shipment by DGC. DGC will replace defective media at no charge to you, provided it is returned postage prepaid to DGC within the ninety (90) day warranty period. This shall be your exclusive remedy and DGC's sole obligation and liability for defective media. This limited media warranty does not apply if the media has been damaged by accident, abuse or misuse.

5. DISCLAIMER OF WARRANTY
EXCEPT FOR THE LIMITED MEDIA WARRANTY NOTED ABOVE, DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS, CLI MACROS OR MATERIALS SUPPLIED HEREUNDER.

6. LIMITATION OF LIABILITY
A. CUSTOMER AGREES THAT DGC'S LIABILITY, IF ANY, FOR DAMAGES, INCLUDING BUT NOT LIMITED TO LIABILITY ARISING OUT OF CONTRACT, NEGLIGENCE, STRICT LIABILITY IN TORT OR WARRANTY SHALL NOT EXCEED THE CHARGES PAID BY CUSTOMER FOR THE PARTICULAR PUBLICATION OR CLI MACRO INVOLVED. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO CLAIMS FOR PERSONAL INJURY CAUSED SOLELY BY DGC'S NEGLIGENCE. OTHER THAN THE CHARGES REFERENCED HEREIN, IN NO EVENT SHALL DGC BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOST PROFITS AND DAMAGES RESULTING FROM LOSS OF USE, OR LOST DATA, OR DELIVERY DELAYS, EVEN IF DGC HAS BEEN ADVISED, KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY THEREOF; OR FOR ANY CLAIM BY ANY THIRD PARTY.

B. ANY ACTION AGAINST DGC MUST BE COMMENCED WITHIN ONE (1) YEAR AFTER THE CAUSE OF ACTION ACCRUES.

7. GENERAL
A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts, excluding its conflict of law rules. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer. DGC hereby rejects all such different, conflicting, or additional terms.

8. IMPORTANT NOTICE REGARDING AOS/VS INTERNALS SERIES (ORDER #1865 & #1875)
Customer understands that information and material presented in the AOS/VS Internals Series documents may be specific to a particular revision of the product. Consequently user programs or systems based on this information and material may be revision-locked and may not function properly with prior or future revisions of the product. Therefore, Data General makes no representations as to the utility of this information and material beyond the current revision level which is the subject of the manual. Any use thereof by you or your company is at your own risk. Data General disclaims any liability arising from any such use and I and my company (Customer) hold Data General completely harmless therefrom.

TIPS ORDERING PROCEDURES

TO ORDER

1. An order can be placed with the TIPS group in two ways:
   a) MAIL ORDER – Use the order form on the opposite page and fill in all requested information. Be sure to include shipping charges and local sales tax. If applicable, write in your tax exempt number in the space provided on the order form.

   Send your order form with payment to: Data General Corporation
   ATTN: Educational Services/TIPS G155
   4400 Computer Drive
   Westboro, MA 01581–9973

   b) TELEPHONE – Call TIPS at (508) 870–1600 for all orders that will be charged by credit card or paid for by purchase orders over $50.00. Operators are available from 8:30 AM to 5:00 PM EST.

METHOD OF PAYMENT

2. As a customer, you have several payment options:
   a) Purchase Order – Minimum of $50. If ordering by mail, a hard copy of the purchase order must accompany order.
   b) Check or Money Order – Make payable to Data General Corporation.
   c) Credit Card – A minimum order of $20 is required for Mastercard or Visa orders.

SHIPPING

3. To determine the charge for UPS shipping and handling, check the total quantity of units in your order and refer to the following chart:

Total Quantity	Shipping & Handling Charge
1–4 Units	$5.00
5–10 Units	$8.00
11–40 Units	$10.00
41–200 Units	$30.00
Over 200 Units	$100.00

If overnight or second day shipment is desired, this information should be indicated on the order form. A separate charge will be determined at time of shipment and added to your bill.

VOLUME DISCOUNTS

4. The TIPS discount schedule is based upon the total value of the order.

Order Amount	Discount
$1–$149.99	0%
$150–$499.99	10%
Over $500	20%

TERMS AND CONDITIONS

5. Read the TIPS terms and conditions on the reverse side of the order form carefully. These must be adhered to at all times.

DELIVERY

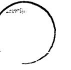
6. Allow at least two weeks for delivery.

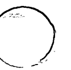
RETURNS

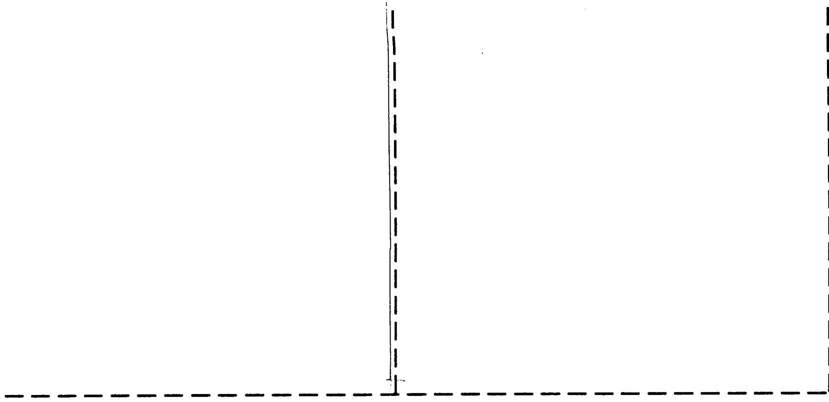
7. Items ordered through the TIPS catalog may not be returned for credit.
8. Order discrepancies must be reported within 15 days of shipment date. Contact your TIPS Administrator at (508) 870–1600 to notify the TIPS department of any problems.

INTERNATIONAL ORDERS

9. Customers outside of the United States must obtain documentation from their local Data General Subsidiary or Representative. Any TIPS orders received by Data General U.S. Headquarters will be forwarded to the appropriate DG Subsidiary or Representative for processing.







Cut here and insert in binder spine pocket

Data General

Data General Corporation, Westboro, Massachusetts 01580