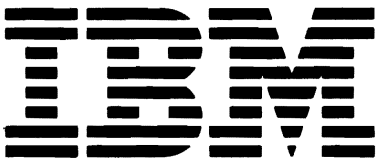
SC21-7741-5

File No. S34-24

IBM System/34
COBOL
Reference Manual

Program Number 5726-CB1



SC21-7741-5

File No. S34-24

IBM System/34
COBOL
Reference Manual

Program Number 5726-CB1

Sixth Edition (July 1985)

This minor revision of SC21-7741 incorporates information relative to the use of the applications described. Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change or addition.

This edition applies to release 9, modification 0 of the IBM System/34 COBOL Program Product (Program 5726-CB1); and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Changes are periodically made to the information herein; these changes will be reported in technical newsletters or in new editions of this publication.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. (For example, ideographic support is available only in Far East countries.) Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Use this publication only for the purposes stated in the *Preface*.

Publications are not stocked at the address below. Requests for copies of IBM publications and for technical information about the system should be made to your IBM representative or to the branch office serving your locality.

This publication could contain technical inaccuracies or typographical errors. Use the Reader's Comment Form at the back of this publication to make comments about this publication. If the form has been removed, address your comments to IBM Canada Laboratory, Information Development, Department 849, Don Mills, Ontario, Canada M3C 1H7. IBM may use and distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

This reference manual describes the System/34 COBOL (Common Business Oriented Language) compiler and language. This manual provides reference material and programmer guide information for persons who have some knowledge of the COBOL language and some experience in writing COBOL programs. The manual is organized as follows:

- Chapters 1 through 5 describe the COBOL language and each of the four divisions: Identification, Environment, Data, and Procedure. The COBOL clauses and statements available to the user are explained.
- Chapter 6 describes the additional functions of the language that are provided through the various processing modules.
- Chapter 7 describes the TRANSACTION file support. A TRANSACTION file allows you to read data from and write data to a display station or an Interactive Communications Feature session.
- Chapter 8 describes the system-dependent considerations.
- Chapter 9 describes how to create, execute, and debug programs.
- Chapter 10 describes the ideographic support provided by COBOL.
- Appendix A contains compiler messages.
- Appendix B describes special purpose subroutines supplied with System/34 COBOL.
- Appendix C contains a COBOL language comparison across various systems.
- Appendix D describes intermediate result fields.
- Appendix E contains sample file-processing programs.
- Appendix F contains COBOL reserved words.
- Appendix G contains the EBCDIC and ASCII collating sequences.
- Appendix H contains a file processing summary and the status key values.
- Appendix I contains a summary of the System/34 COBOL clauses and statements for each division.
- Appendix J contains a summary of display screen format specifications.
- The *Glossary* contains definition of terms.

To aid the user, IBM provides several extensions to ANSI (American National Standards Institute) COBOL, X3.23-1974. The more significant extensions include:

- Extended data types – computational-3 or packed decimal, and computational-4 or binary
- Indexed file support for CORE-INDEX
- Additional debugging support of EXHIBIT and TRACE
- Use of apostrophe instead of quotes

Two methods are available to provide a convenient way to add OCL (Operation Control Language) and procedures to a source library. One method, SEU (Source Entry Utility), is described in the *SEU Reference Manual*. The other method, the use of the job stream command (JOBSTR), is described in the *System Support Reference Manual*. Refer to *Related Publications* in this Preface for the order numbers.

A brief description of the contents of the various System/34 manuals is contained in the *Publications Summary* section of the *IBM System/34 Introduction*.

System Requirements

For information concerning system requirements, refer to the *IBM System/34 Planning Guide*, GC21-5154.

Related Publications

- *IBM System/34 Introduction*, GC21-5153
- *IBM System/34 Bibliography*, GH30-0231
- *IBM System/34 Master Index*, SC21-7739
- *IBM System/34 Functions Reference Manual*, SA21-9243
- *IBM System/34 System Support Reference Manual*, SC21-5155
- *IBM System/34 Operator's Guide*, SC21-5158
- *IBM System/34 Displayed Messages Guide*, SC21-5159
- *IBM System/34 COBOL Reference Summary*, GX21-7746
- *IBM System/34 Source Entry Utility Reference Manual*, SC21-7657
- *IBM System/34 Installation and Modification Reference Manual: Program Products and Physical Setup*, SC21-7689
- *IBM System/34 Overlay Linkage Editor Reference Manual*, SC21-7707
- *IBM System/34 Sort Reference Manual*, SC21-7658
- *IBM COBOL Coding Form*, GX28-1464
- *IBM System/34 1255 Magnetic Character Reader Reference Manual*, SC21-7740
- *IBM System/34 Interactive Communications Feature Reference Manual*, SC21-7751
- *IBM System/34 Concepts and Design Guide*, SC21-7742
- *IBM System/34 Work Station Support Subroutines Reference Manual*, SC21-7810
- *IBM Specifications For Magnetic Character Readers*, GX21-9101
- *IBM Installation Management Manual: An Introduction to Structured Programming in COBOL*, GC20-1776
- *IBM System/34 Screen Design Aid Programmer's Guide and Reference Manual*, SC21-7716

Industry Standards

The System/34 COBOL compiler is designed according to the following industry standards as understood and interpreted by IBM, as of September 1978:

- The ANSI COBOL, X3.23-1974 standard. ANS COBOL is identical to ISO 1989-COBOL, as approved in February 1978 by the International Organization for Standardization. The ANS COBOL processing modules are described in the table under *Language Level* in Chapter 1.
- The December 1975 Federal Information Processing Standard (FIPSPUB 21-1) low-intermediate level. Additional support is provided for many features at higher FIPS levels.

The following are exceptions to the standard:

- No position in a key for an indexed random READ statement or an indexed START statement can be a hex FF (HIGH-VALUE).
- The user must not place a hex FF (HIGH-VALUE) in the first position when using delete-capable files.

Portions of this manual are copied from American National Standards Institute (ANSI) COBOL, X3.23-1974. This material is reproduced with permission from *American National Standard Programming Language COBOL, X3.23-1974*, copyright 1974 by the American National Standards Institute, copies of which may be purchased from the American National Standards Institute at 1430 Broadway, New York, New York, 10018.

Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

Acknowledgment

The following extract from Government Printing Office Form Number 1965-0795689 is presented for the information and guidance of the user:

> Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention COBOL in acknowledgment of the source, but need not quote this entire section.
>
> COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.
>
> No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

Contents

Chapter 1. Introduction

GENERAL DESCRIPTION

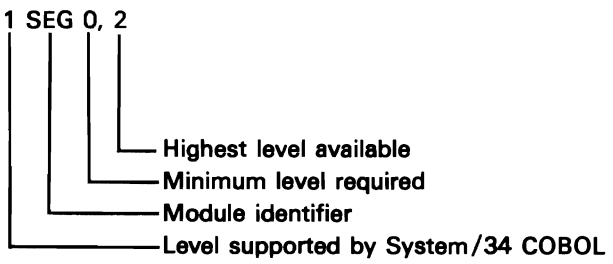
COBOL (Common Business Oriented Language) is a programming language that resembles English. As its name implies, COBOL is especially efficient in the processing of business problems. COBOL can be efficiently used to manipulate large files of data in a relatively simple way. That is, COBOL emphasizes the description and handling of data items and of input/output records.

The System/34 COBOL Compiler and Library is an IBM program product that accepts and compiles COBOL programs written in accordance with the 1974 standard. This program product also includes a number of IBM extensions. The following sections describe the language level implemented and language-independent compiler features.

LANGUAGE LEVEL

The table that follows shows the support of each module provided by System/34 COBOL. The table also describes each module and explains where System/34 COBOL offers more support to a module than is specified by the defined level.

The following example explains the notation used to identify levels of implementation:



System/34 COBOL Processing Modules	Module Description
Nucleus 2 NUC 1, 2	Contains the language elements that are necessary for internal processing.
Table Handling 2 TBL 1, 2	Contains the language elements necessary for: (1) definition of tables; (2) identification, manipulation, and use of indexes; (3) reference to the items within tables. Provides the ability to define fixed-length or variable-length tables of up to three dimensions. Items in the tables can be referred to by using a subscript or an index.
Sequential I/O 2 SEQ 1, 2	Allows definition and access of sequentially organized external files.
Relative I/O 2 REL 0, 2 (see Note)	Provides the capability for defining and accessing disk files in which records are identified by relative record numbers. A file can be accessed randomly and sequentially in the same COBOL program.
Indexed I/O 1 INX 0, 2 (see Note)	Provides the capability for defining disk files in which records are identified by the value of a key and accessed through an index. IBM System/34 COBOL Indexed I/O provides many level 2 functions with two notable exceptions: • ALTERNATE RECORD KEYs are not supported. • The WRITE statement cannot be used when ACCESS is DYNAMIC and the file is opened as I-O.
Note: This module deviates from the standard as described in *Industry Standards* in the *Preface*.	

System/34 COBOL Processing Modules	Module Description
Sort-Merge 2 SRT 0, 2	Allows for the inclusion of one or more sorts in a COBOL program and for use of the merge facility.
Report Writer 0 RPW 0, 1	Provides for semiautomatic production of printed reports.
Segmentation 1 SEG 0, 2	Provides for the overlaying at object time of Procedure Division sections.
Library 2 LIB 0, 2	Allows inclusion of predefined COBOL text into a program.
Debug 1 DEB 0, 2	Provides a means by which the user can specify statements and procedures for debugging.
Inter-program Communication 1 IPC 0, 2	Provides facilities for a program to communicate with one or more other programs. Also provides capability to transfer control to another program known at compile time, and the ability for both programs to have access to certain data items.
Communication 0 COM 0, 2	Provides the ability to access, process, and create messages or portions of messages; also provides the ability to communicate through a Message Control System with local and remote communication devices.

COMPILER FEATURES

The following language-independent features are made available with System/34 COBOL:

- The Diagnosed Source File optionally builds a file that can be retrieved and displayed at a display station. The file that is created contains source statements, merged diagnostics, and summary information. The file can be moved to a library member by an IBM-supplied procedure and can then be reviewed and updated by using SEU.

- Syntax-checking compilation saves machine time while debugging source syntax errors. The source program is scanned for syntax errors and such error messages are generated, but no object code is produced.

- Prompting screen formats provide ease of entry and maintenance by the COBOLP command. This command also allows the specification of parameters needed for compiling and executing COBOL programs.

- The sorted cross-reference option provides a listing of each Data Division name and Procedure Division paragraph name and indicates the statement numbers of each reference of the item.

- Interprogram calls allow programs written in System/34 COBOL to call or be called by other programs written in System/34 COBOL, System/34 FORTRAN IV, or System/34 Basic Assembler.

- Multiple printer files allow the user to define and use multiple printer files in the same program.

- The FIPS (Federal Information Processing Standard) Flagger issues messages identifying nonstandard elements in a COBOL source program. The FIPS Flagger makes it possible to ensure that clauses and statements in a System/34 COBOL source program conform to a particular level of the 1975 Federal Information Processing Standard.

- Diagnostic messages below a user-specified level may be suppressed.

FORMAT NOTATION

In COBOL, basic formats are prescribed for the various elements of the language. In this manual, these formats are presented in a uniform system of notation that is explained in the following paragraphs. This notation is designed to assist the programmer in writing his own COBOL source statements.

- Reserved words are printed entirely in CAPITAL LETTERS. These words have preassigned meanings in COBOL. If any reserved word is misspelled, it is not recognized as a reserved word and may cause an error in the program. The two types of reserved words are key words and optional words.
  - Key words are required by the syntax of the format unless the portion of the format containing them is itself optional. In formats, key words are shown in UNDERLINED CAPITAL LETTERS. If any key word is missing, it is considered an error in the program.
  - Optional words are included only for readability. They may be included or omitted without changing the syntax of the program. Optional words are CAPITALIZED but not underlined.

- Words printed in lowercase letters represent information to be supplied by the programmer. All such words are defined in the text of this manual.

- For easier text reference, some user-defined words are followed by a hyphen and digit or letter. This suffix does not change the syntactical definition of the word.

- Braces ({}) enclosing listed items indicate that one of the enclosed items is required.

- Square brackets ([ ]) indicate that the enclosed item may be used or omitted, depending on the requirements of the program. When two or more items are stacked within brackets, one or none of them may be specified.

- The ellipsis (. . .) indicates that the immediately preceding unit may occur once or any number of times in succession. A unit may be a single lowercase word or a group of lowercase words and one or more reserved words enclosed in brackets and/or braces. When repetition is specified, the entire unit of which the term is a part must be repeated if the term is enclosed within brackets or braces.

- The arithmetic and logical operators (+, -, <, >, =) that appear in formats are required even though they are not underlined.

- All punctuation and other special characters appearing in formats (except braces, brackets, ellipses, commas, and semicolons) are required by the syntax of the format when they are shown; if they are omitted an error occurs in the program. Additional punctuation may be specified, according to the punctuation rules given later in this manual.

- The required clauses and (when written) optional clauses must be written in the sequence shown in the format unless the associated rules explicitly state otherwise.

- Comments, restrictions, and clarifications on the use and meaning of every format are contained in the description that follows each one.

IBM extensions to ANSI COBOL, X3.23-1974, are documented in separate paragraphs beginning with the paragraph heading, **IBM Extension:**

Chapter 2. Language Considerations

COBOL PROGRAM STRUCTURE

Every COBOL source program is divided into four divisions. Each division must be placed in proper sequence, and each must begin with a division header. (Appendix E shows the general structure of every COBOL source program.)

In subsequent chapters, the rules for writing COBOL source programs and methods of data reference are given.

The COBOL Divisions

The four divisions of a COBOL source program and their functions in solving a data processing problem are described in the following paragraphs.

Identification Division

The Identification Division names the program and, optionally, documents the date the program was written, the compilation date, and other pertinent information.

Environment Division

The Environment Division describes the computer(s) to be used and specifies the machine(s) and equipment features used by the program. This description defines the relationship of files of data with input/output devices.

Data Division

The Data Division defines the nature and characteristics of all data the program is to process: the data used in input/output operations and the data developed for internal processing.

Procedure Division

The Procedure Division consists of executable statements that process the data in the manner the programmer defines. Statements are executed in the order they are written unless another order is defined by the programmer.

Clauses and Statements

Every COBOL source program is written in clauses and statements, each of which describes a solution to some specific aspect of the data processing problem.

- Clauses, written in the Environment and Data Divisions, specify an attribute of an entry. A series of clauses ending with a period is defined as an entry.
- Statements, written in the Procedure Division, specify an action to be taken by the object program. A series of statements ending with a period is defined as a sentence.

Each clause or statement in the program can be subdivided into smaller syntactical units called phrases or options. A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL clause or statement. An option is a phrase that provides the programmer with required or optional wording, depending on the desired meaning.

Clauses, entries, statements, and sentences can be combined into paragraphs or sections. Each paragraph and section defines some larger part of the data processing problem solution. Specific rules for the formation of each element are given in the documentation for each division of the COBOL program.

Clause and Statement Specification Order

When specified, each required or optional clause or statement (even those treated as documentation) must be written in the sequence shown in the correct format unless the associated rules explicitly state otherwise.

STRUCTURE OF THE LANGUAGE

In COBOL, the indivisible unit of data is the character. Fifty-one EBCDIC characters form the COBOL character set: the 26 letters of the alphabet, the 10 Arabic numerals, and 15 special characters.

Individual COBOL characters are put together to form character-strings and separators.

A character-string is a character or sequence of contiguous characters that form a word, a literal, a PICTURE character-string, or a comment. A character-string can be delimited only by a separator.

A separator is a contiguous string of one or more punctuation characters. A separator can be placed next to another separator or next to a character-string.

Except for comments and nonnumeric literals (which may use any character within the EBCDIC set), the 51 characters are the only characters valid in a COBOL program. Figure 2-1 shows the valid COBOL characters in ascending EBCDIC sequence and their usage in a COBOL program.

IBM Extension: A System/34 COBOL compiler's default option substitutes an apostrophe (') for a quotation mark ("). Unless the default option is overridden, the quotation mark cannot be used. If conformance with the standard character set is desired, the programmer must specify the quotation mark with a PROCESS statement option at compile time. If the quotation mark is thus specified, the apostrophe cannot be used.

Note: Throughout this manual, the apostrophe is used because it is the default option. In all cases, the quotation mark can be used only if the default option is overridden.

COBOL Character	Meaning	Use
	space	punctuation character
.	decimal point; period	editing character; punctuation character
<	less than	relation character
(	left parenthesis	punctuation character
+	plus symbol	arithmetic operator; sign; editing character
$	dollar sign	editing character
*	asterisk	arithmetic operator; editing character
)	right parenthesis	punctuation character
;	semicolon	punctuation character
-	minus symbol; hyphen	arithmetic operator; sign; editing character
/	stroke or slash	arithmetic operator; editing character
,	comma	punctuation character; editing character
>	greater than	relation character
=	equal sign	relation character; punctuation character
" or '	quotation mark or apostrophe	punctuation character
A-Z	alphabet	alphabetic character
0-9	Arabic numerals	numeric character

Note: All COBOL characters are considered to be alphanumeric.

Figure 2-1. COBOL Characters and Their Meanings

Character-Strings

COBOL character-strings form words, literals, PICTURE character-strings, and comments. Each is described in the following paragraphs.

COBOL Words

A COBOL word can be a user-defined word, a system-name, or a reserved word. A COBOL word can belong to only one of these classes.

The maximum length of a COBOL word is 30 characters.

User-Defined Words: A user-defined word is a COBOL word supplied by the programmer. Valid characters in a user-defined word are:

- A through Z
- 0 through 9
- - (hyphen)

The hyphen may not appear as the first or last character in a user-defined word.

A list of user-defined word sets, together with rules for their formation, is given in Figure 2-2. The function of each user-defined word in any specific clause or statement is included in the prose description for each clause or statement.

User-Defined Word Sets	Rules for Formation
alphabet-name condition-name data-name record-name file-name index-name mnemonic-name routine-name	Must contain at least one alphabetic character. Within each set, the name must be unique either because no other word is made up of an identical character-string, or because it can be made unique through qualification. (See the section *Methods of Data Reference*.)
library-name	Must contain at least one alphabetic character. The system uses the first 8 characters as the identifying name; these first 8 characters, therefore, must be unique among library-names. Text-name must be unique unless qualified by a library-name.
program-name text-name	Must contain at least one alphabetic character. The system uses the first 6 characters as the identifying name. These first 6 characters, therefore, must be unique among program-names.
paragraph-name section-name	Need not contain an alphabetic character. Other rules as in first paragraph.
level-numbers: 01-49, 66, 77, 88	Must be a 1- or 2-digit integer. Need not be unique.
segment-numbers: 00-99	Must be a 1- or 2-digit integer. Need not be unique.

Figure 2-2. User-Defined Word Sets and Rules for Formation

System-Names: A system-name is an IBM-defined name that is used to communicate with the system. A system-name can be:

- computer-name
- language-name
- implementor-name
- function-name

The function of each system-name is described with the format in which it appears; each is defined in the Glossary.

Reserved Words: A reserved word is a COBOL word with fixed meaning(s) in a COBOL source program. A reserved word must not be specified as a user-defined word or as a system-name. Reserved words can be used only as specified in the formats for a COBOL source program.

Appendix F gives a complete list of COBOL reserved words. The section on *Format Notation* in Chapter 1 gives the conventions used to represent reserved words in this manual.

There are six types of reserved words:

- Key words
- Optional words
- Connectives
- Special registers
- Special-character words
- Figurative constants

Each type is described in the following paragraphs.

Key words are words that are required within a given clause, entry, or statement. There are three types of key words:

- Verbs, such as ADD, READ, WRITE
- Required words, which appear in clause, entry, or statement formats, such as the word USING in the MERGE statement
- Words with a specific functional meaning, such as NEGATIVE or SECTION

Optional Words are words that may be included in a clause, entry or statement. When an optional word is omitted, the meaning of the COBOL program is unchanged.

There are three types of *connectives*: qualifier, series, or logical.

- Qualifier connectives (OF, IN) associate a data-name, condition-name, text-name, or paragraph-name with its qualifier.
- Series connectives (the comma and semicolon) optionally link two or more consecutive operands. (An operand is a data item or literal that is acted upon by the COBOL program.)
- Logical connectives (AND, OR, AND NOT, OR NOT) are used in specifying conditions.

Special registers are compiler-generated storage areas used primarily to store information produced through one of the specific COBOL features. Each such storage area has a fixed name and need not be further defined within the program. These special registers include the following:

- DEBUG-ITEM (See *Debugging Features* in Chapter 6.)
- LINAGE-COUNTER (See *LINAGE Clause* in Chapter 4.)
- DATE, DAY, TIME (See ACCEPT Statement in Chapter 5.)

Special-character words are arithmetic operators (+ - / * **) or relation characters (< > =). Arithmetic operators are described under *Arithmetic Expressions* in Chapter 5. Relation characters are described in the relation condition description of the *Conditional Expressions* in Chapter 5.

Figurative constants name and refer to specific constant values.

The reserved words for figurative constants and their meanings are:

- ZERO, ZEROES, ZEROS–represents the value 0 or one or more occurrences of the character 0, depending on context. Zero can be numeric or nonnumeric, depending on context. For example, ZERO is considered to be nonnumeric when it is used in a relational expression in which it is compared to an alphanumeric data item.

- SPACE, SPACES–represents one or more blanks or spaces. Must be nonnumeric.

- HIGH-VALUE, HIGH-VALUES–represents one or more occurrences of the character that has the highest value in the collating sequence used. For the EBCDIC (NATIVE) collating sequence, the character is hex FF; for other collating sequences, the character used depends on the collating sequence. When used in a COBOL program, HIGH-VALUE is treated as a nonnumeric literal.

- LOW-VALUE, LOW-VALUES–represents one or more occurrences of the character with the lowest value in the collating sequence used. For the EBCDIC (NATIVE) collating sequence, the character is hex 00; for other collating sequences, the character used depends on the collating sequence. When used in a COBOL program, LOW-VALUE is treated as a nonnumeric literal.

- QUOTE, QUOTES–represents one or more occurrences of the quotation mark character and must be nonnumeric. The word QUOTE (QUOTES) cannot be used in place of an apostrophe to enclose a nonnumeric literal.

- ALL literal–represents one or more occurrences of the string of characters composing the literal and must be nonnumeric. The literal must be either a nonnumeric literal or a figurative constant other than the ALL literal. When a figurative constant is used, the word ALL is redundant and is used for readability only. The figurative constant ALL literal must not be used with the DISPLAY, INSPECT, STRING, STOP, or UNSTRING statements.

The singular and plural forms of a figurative constant are equivalent and can be used interchangeably. For example, if DATA-NAME-1 is a five-character data item, either of the following statements will fill DATA-NAME-1 with five spaces:

```
MOVE SPACE TO DATA-NAME-1.
MOVE SPACES TO DATA-NAME-1.
```

In any format, a figurative constant may be substituted for a nonnumeric literal; only the figurative constant ZERO (ZEROS, ZEROES) may be substituted for a numeric literal.

The length of a figurative constant depends on the context of the program. The following rules apply:

- When a figurative constant is associated with a data item, the length of the figurative constant character-string is equal to the length of the associated data item. This rule applies, for example, when a figurative constant is moved to or compared with another item.

- When a figurative constant is not associated with another data item, the length of the character-string is one character. This rule applies, for example, in the DISPLAY, INSPECT, STRING, STOP, and UNSTRING statements.

Literals

A literal is a character-string with a value that is specified either by the ordered set of characters of which it is composed or by a figurative constant. The three types of literals are nonnumeric, numeric, and Boolean.

Nonnumeric Literals: A nonnumeric literal is a character-string that can contain any allowable character from the EBCDIC set. A nonnumeric literal may contain a maximum of 120 characters.

A nonnumeric literal must be enclosed by apostrophes. The enclosing apostrophes are not part of the literal.

Any punctuation characters included within a nonnumeric literal are part of the value of the literal. An embedded apostrophe must be represented by a pair of adjacent apostrophes; one apostrophe is then part of the value of the literal. Each pair of embedded apostrophes in the literal counts as one character against the limit of 120 characters.

Every nonnumeric literal is in the alphanumeric category. Data categories are defined under *PICTURE Clause* in Chapter 4.

Numeric Literals: A numeric literal is a character-string whose characters are selected from the digits 0 through 9, a sign character (+ or -), and/or the decimal point. The following rules apply:

- One to 18 digits are allowed.
- Only one sign character is allowed. If a sign character is included, it must be the leftmost character of the literal. If the literal is unsigned, it is considered to have a positive value.
- Only one decimal point is allowed. If a decimal point is included, it is treated as an assumed decimal point (not considered a character position in the literal). The decimal point may appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, it is considered to be an integer. The word integer appearing in a format represents a numeric literal of nonzero value that contains no sign and no decimal point; any other restrictions are included with the description of the format.

The value of a numeric literal is the algebraic quantity expressed by the characters in the literal. The size of a numeric literal in standard data format characters is equal to the number of digits specified by the user.

IBM Extension:

Boolean Literals: A Boolean literal contains a single 0 or 1 and is enclosed in quotes and immediately preceded by an identifying B. The Boolean literal is defined as either B'0' or B'1'. A Boolean character occupies one byte. The figurative constant ZERO can be used as a Boolean literal, and the reserved word ALL is valid with a Boolean literal.

PICTURE Character-Strings

A PICTURE character-string consists of COBOL characters used as symbols in the PICTURE clause. The choice of symbols determines whether the user-defined name is Boolean, numeric, alphabetic, or alphanumeric, and is also used to define edited output fields.

Comments

A comment is a character-string containing any combination of characters from the EBCDIC set. A comment serves only as documentation. Comments take two forms:

- A comment-entry in the Identification Division. For a further description of a comment entry, see *Identification Division* in Chapter 3.
- A comment line (preceded by an asterisk or a slash in Column 7) in any division of the program. For a further description of a comment line, see *Standard COBOL Format* in this chapter.

Separators

A separator is a string of one or more punctuation characters. The punctuation characters are shown in Figure 2-3.

Punctuation Character	Meaning
	space
.	period
(	left parenthesis
)	right parenthesis
;	semicolon
,	comma
=	equal sign
"	quotation mark
'	apostrophe

Figure 2-3. Punctuation Characters

The following rules apply to the formation of separators:

- A space is always a separator except when the space appears within a nonnumeric literal. When contained between the opening and closing apostrophe of a nonnumeric literal, the space is considered part of the literal. Wherever a space is used as a separator, more than one space can be used.
- A comma, semicolon, or period that is immediately followed by a space is a separator. These separators may appear only where explicitly allowed by COBOL rules.
- The left and right parentheses are separators. Parentheses must appear as balanced pairs of left and right parentheses, delimiting subscripts, indexes, arithmetic expressions, or conditions.
- The apostrophe is a separator. An opening apostrophe must be immediately preceded by a space or a left parenthesis. A closing apostrophe must be immediately followed by one of the following separators: space, comma, semicolon, period, or right parenthesis. Apostrophes must appear as balanced pairs delimiting nonnumeric literals except when the literal is continued.
- The pseudo-text delimiter (==) is a separator. An opening pseudo-text delimiter must be immediately preceded by a space. A closing pseudo-text delimiter must be immediately followed by one of the following separators: space, comma, semicolon, or period. Pseudo-text delimiters must appear as balanced pairs delimiting pseudo-text.

STANDARD COBOL FORMAT

COBOL programs must be written in the standard COBOL format, described in the following discussion. The format is described in terms of an 80-character line. The output listing of the source program is printed in this same format. The COBOL coding form is shown in Figure 2-4.

Sequence Numbers (Columns 1-6)

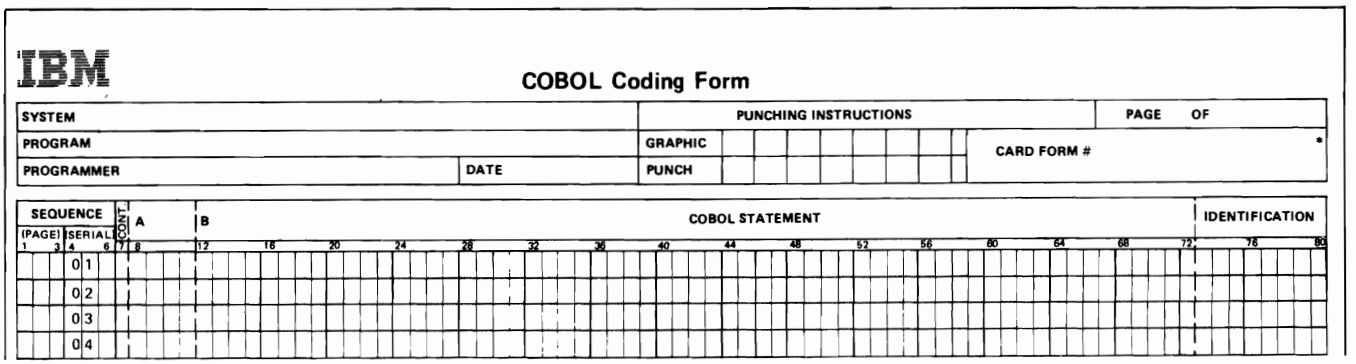
Sequence numbers are written in columns 1 through 6. A sequence number is used to identify numerically each line to be compiled by the COBOL compiler. The use of sequence numbers is optional. A sequence number, if used, must consist of six digits in the sequence number area (including the preprinted digits in columns 4 and 5).

If sequence numbers are present in the source program, they must be in ascending order. If sequence numbers are out of sequence, the compiler accepts them in the order read and generates a warning message.

IBM Extension: The user may suppress sequence checking at compile time.

Continuation Area (Column 7)

The continuation area is used to indicate the continuation of words and nonnumeric literals from the previous line onto the current line, to specify debugging lines, or to indicate that the text on this line is to be treated as a comment.



Columns 1-6 represent the sequence number area.
Column 7 is the continuation area.
Columns 8-11 represent Area A } Used for writing COBOL source statements.
Columns 12-72 represent Area B }
Columns 73-80 are used to identify the program.

Figure 2-4. IBM COBOL Coding Form and Standard COBOL Format

Area A (Columns 8-11) and Area B (Columns 12-72)

COBOL elements that may begin in Area A and specific COBOL elements that may follow them are shown in Figure 2-5.

Elements That Begin in Area A	Must be Followed Immediately By	Placement of Following Elements
Division Header	USING Option	Same or next line (Area B)
	Section header, paragraph header, paragraph-name, or (in Procedure Division) key word DECLARATIVES	Next line (Area A)
Section Header	COPY or USE statement	Same or next line (Area B)
	Paragraph header or paragraph-name (after COPY or USE, if specified)	Next line (Area A)
Paragraph Header or Paragraph-Name	Environment Division entry or Procedure Division sentence	Same or next line (Area B)
Level Indicator Level-number	Data-name	Same line (Area B)
Key Word DECLARATIVES	Declaratives section name	Next line (Area A)
Key Words END DECLARATIVES	Section-header	Next line (Area A)

Figure 2-5. Sequence of Elements in Area A and Area B

The basic skeleton of a COBOL program is shown in Figure 2-6.

```
SEQUENCE      CONT  A       B
PAGE SERIAL
001   01            IDENTIFICATION DIVISION.
      02
      03
      04            ENVIRONMENT DIVISION.
      05            CONFIGURATION SECTION.
      06
      07            INPUT-OUTPUT SECTION.
      08            FILE-CONTROL.
      09
      10            DATA DIVISION.
      11            FILE SECTION.
      12            FD
      13            WORKING-STORAGE SECTION.
      14            77  DESCRIPTION
      15            01
      16
      17            PROCEDURE DIVISION.
      18            DECLARATIVES.
      19
      20            END DECLARATIVES.
                    SECTION-NAME SECTION.
                    PARAGRAPH-NAME.
                *   COMMENTS.
                D   DEBUG-STATEMENTS.
                    MOVE 'TEST' TO A.
```

Figure 2-6. Basic Skeleton of a COBOL Program

Special Considerations

Some lines in a COBOL program require additional rules. The divisional headers and the Data Division entries have special requirements. A discussion of each follows.

Division Header

A division header must be immediately followed by a period except when a USING option is specified with a Procedure Division header. Except for the USING option, no text may appear on the same line.

Section Header

A section header must be immediately followed by a period except when Procedure Division segment numbers are specified. In the Environment and Procedure Divisions, a section consists of paragraphs. In the Data Division, a section consists of Data Division entries.

Paragraph Header, Paragraph-Name

In the Environment Division, a paragraph consists of a paragraph header followed by one or more entries in Area B. An entry consists of one or more clauses. In the Procedure Division, a paragraph consists of a paragraph-name followed by one or more sentences in Area B. A sentence consists of one or more statements; a statement is a syntactically valid combination of a COBOL verb and its operands. Entries and sentences must be ended with a period followed by a space.

Successive entries or sentences begin in Area B. The entries are either on the same line as the last entry or sentence, or they are on the next succeeding nonblank noncomment line.

Data Division Entries

Each Data Division entry begins with a level indicator or level-number followed by a space. On the same line is a data-name in Area B, followed by a sequence of independent clauses describing the item. Each clause, except the last, is followed by a space (or optionally by a comma or semicolon and a space). The last clause in the entry must be ended with a period followed by a space.

Successive clauses begin in Area B. The clauses are either on the same line as the preceding clause, or on the next succeeding nonblank noncomment line.

A level indicator (FD, SD) must begin in Area A and be followed by a space. For a further description of level indicators, see *Data Division Organization* in Chapter 4.

A level number is a one- or two-digit integer with one of the following values: 1 through 49, 66, 77, or 88. At least one space must follow the level-number.

Level-numbers 01 and 77 must begin in Area A. The associated record-name or item-name must appear in Area B. Level-numbers 02 through 49, 66, and 88 may begin in either Area A or Area B.

DECLARATIVES and END DECLARATIVES

In the Procedure Division, the key words DECLARATIVES and END DECLARATIVES begin and end the declaratives portion of the source program. Both of these key words must begin in Area A and be followed immediately by a period. No other text may appear on the same line. After the key word END DECLARATIVES, no text may appear before the following section header.

Program Spacing

In writing a COBOL program, rules for indentation, continued lines, comment lines, debugging lines, and blank lines must be observed.

Indentation

Within an entry or sentence, successive lines in Area B may have the same format or may be indented to clarify program logic. The output listing is indented only if the input statements are indented. Indentation does not affect the syntax of the program. The amount of indentation can be chosen by the programmer, subject only to the restrictions on the width of Area B.

Continuation of Lines

Any sentence, entry, clause, or phrase that requires more than one line can be continued in Area B of the next succeeding noncomment line. The line being continued is called the continued line; the succeeding lines are continuation lines. Area A of a continuation line must contain only spaces.

If there is no hyphen in the continuation area (Column 7) of a line, the last character of the preceding line is assumed to be followed by a space.

If there is a hyphen in the continuation area of a line, the first nonblank character of this continuation line immediately follows the last nonblank character of the continued line without any intervening space. However, this restriction does not apply to nonnumeric literals.

If the continued line contains a nonnumeric literal without a closing apostrophe, all spaces at the end of the continued line (through Column 72) are considered to be part of the literal. The continuation line must contain a hyphen in the continuation area, and the first nonblank character in Area B must be an apostrophe. The continuation of the literal begins with the character immediately following the apostrophe.

Comment Lines

A comment line is any line with an asterisk or slash in the continuation area of the line. The comment may be written anywhere in Area A and Area B of that line. The comment may consist of any combination of characters from the EBCDIC set.

If an asterisk is placed in the continuation area, this comment line is printed in the output listing immediately following the last preceding line.

If the slash is placed in the continuation area, the current page of the output listing is ejected, and the comment line is printed on the first line of the next page.

The asterisk or slash and the comment are produced only on the output listing. They are treated as documentation by the compiler.

Successive comment lines are allowed. Each must begin with an asterisk or slash in the continuation area.

Debugging Lines

A debugging line is any line with a D coded in the continuation area. Rules for the formation of debugging lines are given under *Debugging Features* in Chapter 6.

Blank Lines

Blank lines contain nothing but spaces from Column 7 through Column 72. A blank line may appear anywhere in a program except immediately preceding a continuation line.

Overall Punctuation Rules

Any punctuation character included in a PICTURE character-string, a comment character-string, or a nonnumeric literal is not considered to be a punctuation character but rather is considered to be part of the character-string or literal.

A comma, period, or semicolon followed by a space in or at the end of a PICTURE character-string is a separator and will terminate the PICTURE character-string. The comma and semicolon are used only for readability.

Punctuation rules for each division of the COBOL source program follow.

Identification Division

Commas and semicolons can be used in the comment-entries. The PROGRAM-ID paragraph must end with a period followed by a space.

Environment Division

Commas or semicolons may separate successive clauses and successive operands within clauses. The SOURCE-COMPUTER, OBJECT-COMPUTER, SPECIAL-NAMES, and I-O-CONTROL paragraphs must each end with a period followed by a space. In the FILE-CONTROL paragraph, each file-control entry must end with a period followed by a space.

Data Division

Commas or semicolons may separate successive clauses and operands within clauses. File (FD), Sort/Merge file (SD), and data description entries must each end with a period followed by a space.

Procedure Division

Commas or semicolons may separate successive statements within a sentence and successive operands within a statement. Each sentence and each procedure must end with a period followed by a space.

METHODS OF DATA REFERENCE

Every user-specified name defining an element in a COBOL program must be unique, either because no other name has a character-string of the same value or because it can be made unique through qualification, subscripting, or indexing. In addition, references to data and procedures can be either explicit or implicit. The rules for qualification and for explicit and implicit references follow.

Qualification

A name can be made unique if it exists within a hierarchy of names, and the name can be identified by specifying one or more higher-level names in the hierarchy. The higher-level names are called qualifiers, and the process by which such names are made unique is called qualification.

Qualification is specified by placing one or more phrases after a user-specified name. Each phrase consists of the word OF or IN followed by a qualifier. (OF and IN are logically equivalent.) The three formats for references are references to Data Division names, references to Procedure Division names, and references to COPY libraries.

Format 1

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

Format 2

```
paragraph-name [ { OF } section-name ]
               { IN }
```

Format 3

```
text-name [ { OF } library-name ]
          { IN }
```

In Data Division references, all qualifying data-names must be associated with a level-indicator or level-number. Therefore, two identical data-names must not appear as subordinate entries in a group item unless they can be made unique through qualification. Names associated with a level-indicator (FD and SD) are the highest level in the hierarchy. Next highest are those associated with level-number 01. Names associated with level-numbers 02 through 49 are at successively lower levels in the hierarchy.

In the Procedure Division, two identical paragraph-names must not appear in the same section. A section-name is the highest and only qualifier available for a paragraph-name.

The following example illustrates the use of identical names in a data hierarchy:

```
01  FIELD-A
    02  FIELD-B
        05  SUB-1
            07  SUB-2
    02  FIELD-C
        05  SUB-1
    02  FIELD-D REDEFINES FIELD-C
        05  SUB-3
        05  SUB-4
```

A hierarchy includes all subordinate entries to the next equal or higher level number. Therefore, in the above example all entries are in the hierarchy of FIELD-A. All entries from FIELD-B to but not including FIELD-C are in the hierarchy of FIELD-B.

In the hierarchy of FIELD-A, SUB-1 can be used twice; once as subordinate to FIELD-B and once as subordinate to FIELD-C. When referring to SUB-1, it must be qualified as SUB-1 OF FIELD-B or SUB-1 OF FIELD-C. Within FIELD-B or FIELD-C, SUB-1 cannot be subordinate to itself.

SUB-2, SUB-3, and SUB-4 can also be qualified when they are referenced. This qualification is optional, as these fields are already uniquely identified.

In any hierarchy, the name associated with the highest level must be unique and cannot be qualified. No matter what qualification is available, no name can be both a data-name and a procedure-name.

Enough qualification must be specified to make the name unique; however, it may not be necessary to specifiy all the levels of the hierarchy. For example, if more than one file has records that contain the field EMPLOYEE-NO but only one of the files has a record named MASTER-RECORD, then specifying EMPLOYEE-NO OF MASTER-RECORD sufficiently qualifies EMPLOYEE-NO. EMPLOYEE-NO OF MASTER-RECORD OF MASTER-FILE is valid but unnecessary.

Qualification Rules

The following rules for qualification apply:

- Each qualifier must be of a successively higher level and must be within the same hierarchy as the name it qualifies.
- The same name must not appear at two levels in a hierarchy unless it can be qualified.
- If a data-name or condition-name is assigned to more than one data item, the data item must be qualified each time it is referenced. This rule has one exception; the condition-name must not be qualified when used in a REDEFINES clause.
- A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to within the section in which it appears.
- Library-name must be unique in the system. Therefore, the first eight characters of library-name must be unique.
- Text-name must be qualified by the library-name of the library in which it resides, reside in the library specified on the LIBRARY option of the PROCESS statement, or reside in the system library if no other qualification is given.
- When a data-name is being used as a qualifier, it cannot be subscripted.
- A name can be qualified even when it does not need qualification.

- If more than one combination of qualifiers ensures uniqueness, then any of these combinations can be used.
- Duplicate section-names are not allowed.
- A data-name cannot be the same as a section-name or a paragraph-name.
- If a data-name cannot be made unique by qualification, duplication of this data-name is not allowed.
- The complete list of qualifiers for one data-name must not be the same as a partial list of qualifiers for another data-name.

Subscripting and Indexing

Subscripts and indexes can be used only when reference is made to an individual element within a table of elements that have not been assigned individual data-names. Subscripting and Indexing are explained under *Table Handling* in Chapter 6.

Identifier

An identifier is a term used to reflect that a data-name, if not unique in a program, must be followed by a syntactically correct combination of qualifiers, subscripts, or indices necessary to ensure uniqueness. The general formats for identifiers are as follows:

Format 1

```
data-name-1 [ { OF } data-name-2 ] ... [ (subscript-1 [,subscript-2 [,subscript-3]]) ]
            [ { IN }             ]
```

Format 2

```
data-name-1 [ { OF } data-name-2 ] ... [ ( { index-name-1 [ {±} literal-2 ] } )
            [ { IN }             ]     [ { literal-1                       } ]

[ , { index-name-2 [ {±} literal-4 ] } ] [ , { index-name-3 [ {±} literal-6 ] } ] ] ) ]
  { literal-3                       }     { literal-5                       }
```

Restrictions on qualification, subscripting, and indexing follow:

- A data-name must not be subscripted or indexed when that data-name is being used as an index, subscript or qualifier.
- Indexing is not permitted when subscripting is not permitted.
- An index can be modified only by the SET, SEARCH, and PERFORM statements. Data items described by the USAGE IS INDEX clause permit the values associated with index-names to be stored as a binary occurrence number. Such data items are called index data items.
- Literal-1, literal-3, literal-5 in the above format must be positive numeric integers. Literal-2, literal-4, literal-6 must be unsigned numeric integers.

Condition-Name

A condition-name is a user-defined word that is assigned a specific value or range of values. The value assigned is contained in the set of values that a conditional variable may possess. A condition-name can alternatively be a user-defined word that is assigned the status of an IBM-supplied switch or device.

Each condition-name must be unique, or it must be made unique through qualification, and/or indexing, or subscripting.

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

If references to a conditional variable require indexing or subscripting, then references to any of its condition-names also require the same combination of indexing or subscripting.

The format and restrictions on the combined use of qualification, subscripting, and indexing of condition-names are the same as those for identifiers except that data-name-1 is replaced by condition-name-1.

In the general formats, condition-name refers to a condition-name that is qualified, indexed, or subscripted as necessary.

Explicit and Implicit References

COBOL source program references can be either explicit or implicit in three instances: data attribute specification, Procedure Division data references, and transfers of control.

Data Attribute Specification

Explicit attributes are specified in COBOL coding. If a data attribute is not an explicit attribute (not specified in COBOL coding), it takes on a default value. These default values are implicit attributes.

For example, the ACCESS MODE clause in the file-control entry need not be specified. If the clause is omitted, the compiler provides the default value, ACCESS MODE IS SEQUENTIAL. This clause is then an implicit attribute. If this same attribute, ACCESS MODE IS SEQUENTIAL, is specified in the COBOL coding, it is an explicit attribute.

Procedure Division Data References

Procedure Division statements can refer to data items either explicity or implicitly.

An explicit reference occurs when the data-name of the item is written in a COBOL statement or when the data-name is copied into the program through a COPY statement. An implicit reference occurs when the data-name is referred to by a COBOL statement without the name being written in that statement.

For example, when a USE AFTER STANDARD EXCEPTION/ERROR PROCEDURE ON INPUT is specified, an implicit reference is made to each file-name that identifies an input file. For a further description, see *EXCEPTION/ERROR Declarative* in Chapter 5.

TRANSFERS OF CONTROL

In the Procedure Division, program flow transfers control from statement to statement in the order they are written unless an explicit control transfer is specified or no next executable statement exists. (See note below.) This normal program flow is an implicit transfer of control.

In addition to the implicit transfers of control between consecutive statements, implicit transfer of control also occurs when the normal flow is altered without the execution of a procedure branching statement. COBOL provides implicit transfers of control that override the statement-to-statement transfers of control under the following conditions:

- After execution of the last statement of a procedure being executed under control of another COBOL statement. COBOL statements that control procedure execution are MERGE, PERFORM, SORT, and USE.
- During SORT or MERGE statement execution when control is transferred to any input or output procedure.
- During execution of any COBOL statement that causes execution of a Declarative procedure.
- At the end of execution of any Declarative procedure.

COBOL also provides explicit transfers of control through the execution of a procedure branching or conditional statement. Lists of procedure branching and conditional statements are given under *Procedure Division Organization* in Chapter 5.

Note: The term *next executable statement* refers to the next COBOL statement to which control is transferred according to the rules given above. No next executable statement can follow:

- The last statement in a Declarative procedure that is not being executed under control of another COBOL statement.
- The last statement in a COBOL program when the procedure in which it appears is not being executed under control of another COBOL statement.

Chapter 3. Identification and Environment Divisions

Identification Division

The Identification Division must be the first division in every COBOL source program. This division names the object program. (An object program is the output from a compilation.)

The user may also include the date the program was written, the date of compilation, and other such documentary information about the program in the Identification Division.

Format

```
IDENTIFICATION DIVISION.

PROGRAM-ID. program-name.

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

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

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

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

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

The Identification Division must begin with the words IDENTIFICATION DIVISION followed by a period and a space.

Coding Example

```
SEQUENCE
001 010 IDENTIFICATION DIVISION.
    020 PROGRAM-ID. SAMPLE.
    030 AUTHOR. A PROGRAMMER.
    040 INSTALLATION. ROCHESTER LAB.
    050 DATE-WRITTEN. 04/11/79.
    060 DATE-COMPILED. 08/16/79.
    070 SECURITY. NON-CONFIDENTIAL.
```

PROGRAM-ID Paragraph

The first paragraph of the Identification Division must be the PROGRAM-ID paragraph. The PROGRAM-ID paragraph specifies the name by which the object program is known to the system.

Program-name is a user-defined word that identifies the object program to the system. A program-name must include at least one alphabetic character. The system uses the first 6 characters of program-name as the identifying name of the program; these first 6 characters, therefore, should be a unique program-name.

The system expects the first character of program-name to be alphabetic; if it is numeric, it is converted as follows:

- 0 is converted to J
- 1-9 is converted to A-I

The system does not allow the hyphen as a program-name character; therefore, if the hyphen is the second through sixth character, it is converted to zero.

To avoid such conversions, the programmer should not specify program-names with leading numerics or embedded hyphens.

Other Optional Paragraphs

The other paragraphs are optional; however, if they are written, they must appear in the order shown in the format.

The comment-entries serve only as documentation and do not affect the syntax of the program. The comment-entries in the optional paragraphs may be any combination of characters from the EBCDIC set and may be written in Area B on one or more lines. A hyphen is not permitted in the continuation area of Identification Division statements.

The DATE-COMPILED paragraph provides the compilation date of the source listing. When the comment-entry is specified, the entire entry is replaced with the current date. When the comment-entry is omitted, the compiler adds the current date to the line on which DATE-COMPILED is printed.

Environment Division

The Environment Division, the second division of all COBOL source programs, identifies the following:

- The computer on which the source program is to be compiled
- The computer on which the object program is to be executed
- The specific main storage size required to execute the object program
- The linkage between the logical concept of the files and their records, and the physical aspects of the devices on which data is stored

The Environment Division has two sections: the Configuration Section and the Input-Output Section.

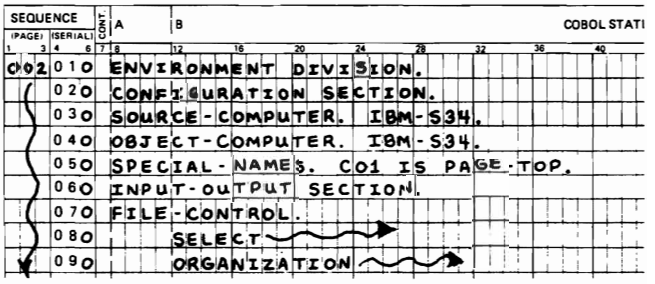
The following shows the general format of the sections and paragraphs in the Environment Division, and defines the order of presentation in the source program.

Format

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. source-computer-entry

OBJECT-COMPUTER. object-computer-entry

[SPECIAL-NAMES. special-names-entry]

[INPUT-OUTPUT SECTION.

FILE-CONTROL. {file-control-entry} . . .

[I-O-CONTROL. input-output-control-entry]] .
```

The Environment Division must begin with the words ENVIRONMENT DIVISION followed by a period and a space.

Coding Example



CONFIGURATION SECTION

The Configuration Section describes the computer that compiles the source program and the computer that executes the object program. This section optionally relates IBM-defined function names to user-defined mnemonic-names, specifies the collating sequence to be used, specifies a substitution for the currency sign, or interchanges the functions of the comma and the period.

In the Configuration Section, the comma or semicolon can optionally separate successive clauses within a paragraph. In each paragraph, there must be one period; the period must be placed immediately after the last entry in the paragraph.

Format

```
CONFIGURATION SECTION.

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

OBJECT-COMPUTER. computer-name

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

    [,PROGRAM COLLATING SEQUENCE IS alphabet-name]

    [,SEGMENT-LIMIT IS segment-number] .

[ SPECIAL-NAMES. [, function-name-1 IS mnemonic-name] ...

    [function-name-2

    { IS mnemonic-name,ON STATUS IS condition-name-1 [,OFF STATUS IS condition-name-2] }
    { IS mnemonic-name,OFF STATUS IS condition-name-2 [,ON STATUS IS condition-name-1] }
    { ON STATUS IS condition-name-1 [, OFF STATUS IS condition-name-2] }
    { OFF STATUS IS condition-name-2 [, ON STATUS IS condition-name-1] } ] ...

    [ ,alphabet-name IS { STANDARD-1 }
                        { NATIVE     }
                        { literal-1 [ { THROUGH } literal-2                                  ] }
                        {             { THRU    }                                            }
                        {             ALSO literal-3 [,ALSO literal-4] ... ]                 }
                        {                                                                    }
                        {   [ literal-5 [ { THROUGH } literal-6                          ] ] ... }
                        {               { THRU    }                                      }   }
                        {               ALSO literal-7 [ ,ALSO literal-8 ] ...           }   } ] ...

    [, CURRENCY SIGN IS literal-9 ]

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

SOURCE-COMPUTER Paragraph

The SOURCE-COMPUTER paragraph describes the computer that compiles the source program. The computer-name should be coded as: IBM-S34. If the computer-name is not coded, it is assumed to be IBM-S34.

With the exception of the WITH DEBUGGING MODE clause, the SOURCE-COMPUTER paragraph is treated as documentation. The WITH DEBUGGING MODE clause is described under *Debugging Features* in Chapter 6.

OBJECT-COMPUTER Paragraph

The OBJECT-COMPUTER paragraph identifies the computer that executes the object program. Computer-name should be the first entry in the OBJECT-COMPUTER paragraph. The computer-name should be coded as: IBM-S34. If the computer-name is not coded, it is assumed to be IBM-S34.

Except for the PROGRAM COLLATING SEQUENCE and MEMORY-SIZE clauses, the OBJECT-COMPUTER paragraph is treated as documentation.

MEMORY SIZE Clause

The MEMORY SIZE clause can be used to specify the amount of main storage required to execute the object program. When the LINK option is specified on the PROCESS statement, the value specified for the MEMORY SIZE clause is passed to the overlay linkage editor. If the MEMORY SIZE clause specifies a region larger than the unoverlayed size of the object program, the region size specified is the amount of main storage that is assigned to the program. When the MEMORY SIZE clause specifies a region that is not large enough to contain the object program, overlays are created in the attempt to fit the program into the size specified. If the program cannot be successfully overlayed, a message is displayed giving the operator the choice of not link-editing or producing a load module that is not overlayed. It is necessary to allow enough memory space to compile and link-edit the program. The Overlay Linkage Editor takes the compiler region size (18K) as the default for the size of region to link the program, unless:

- A value is placed in the memory size clause.
- A // REGION statement is issued before compilation.
- The default value for the work station region size is changed by means of the SET command.

If the size thus determined is not sufficient, a message (SYS–3172) issued by the Overlay Linkage Editor causes the Overlay linkage editor to increase (if possible) the region size to allow the link edit.

Note: If the total size of the linked program exceeds 64K, the program must then be segmented to allow overlaying. (For more information on the region size, see *REGION Statement* in the *SSP Reference Manual.)*

Regardless of whether WORDS, CHARACTERS, or MODULES are coded, the value of integer is interpreted by the compiler as bytes or character spaces.

PROGRAM COLLATING SEQUENCE Clause

The PROGRAM COLLATING SEQUENCE clause specifies the collating sequence used in a program. The collating sequence associated with the specified alphabet-name must be defined in the SPECIAL-NAMES paragraph. The program collating sequence is used to determine the truth value of the following nonnumeric comparisons:

- Those comparisons explicitly specified in relation conditions.
- Those comparisons explicitly specified in condition-name conditions.

The PROGRAM COLLATING SEQUENCE clause also applies to any nonnumeric merge or sort keys unless the COLLATING SEQUENCE option is specified in the MERGE or SORT statement.

When the PROGRAM COLLATING SEQUENCE clause is omitted, the EBCDIC collating sequence is used. See Appendix G for the complete EBCDIC collating sequence.

SEGMENT-LIMIT Clause

Segment-number must be an integer ranging in value from 1 through 49. Segment-number is treated as comments. If specified, it must be correct.

SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph relates IBM-specified function-names to user-specified mnemonic-names. This paragraph specifies a collating sequence that is associated with an alphabet-name, a substitute character for the currency sign, and the interchange of the comma and decimal point in PICTURE clauses and numeric literals.

Function-Name-1 Clause

Function-name-1 specifies system devices or standard system actions taken by the compiler.

The associated mnemonic-name is required. The mnemonic-name is formed according to the rules for a user-defined word and is required to contain at least one alphabetic character.

Figure 3-1 shows the actions that are associated with mnemonic-names for function-name-1. Each of these functions may appear only once in the SPECIAL-NAMES paragraph.

Function-Name-1	Action
SYSTEM-CONSOLE	When specified, the associated mnemonic-name can be used in ACCEPT and DISPLAY statements to communicate with the system console.
REQUESTOR	When specified, the associated mnemonic-name can be used in ACCEPT and DISPLAY statements to communicate with the user display station.
CSP	Suppress spacing after printing a line.
C01	Skip to next page.
LOCAL-DATA	When specified, the associated mnemonic-name can be referenced by either an ACCEPT or a DISPLAY statement that references a display station or MRT procedure. An ACCEPT or a DISPLAY statement is issued to retrieve data from, or store data in, a system-managed area that provides communications with programs that are executed sequentially within a display station job. The LOCAL-DATA area is described in the chapter on OCL statements under the LOCAL statement in the *System Support Reference Manual*.
ATTRIBUTE-DATA	When specified, the associated mnemonic-name can be referenced only in an ACCEPT statement. The reference causes an attribute record that is associated with an identified display station or SSP-ICF session to be input to the data item coded in the ACCEPT statement. Attribute records and their required formats are described in Chapter 7.

Figure 3-1. Choices of Function-Name-1 and Action Taken

Function-Name-2 Clause

Function-name-2 defines a one-byte program switch. Function-name-2 can be defined as UPSI-0 through UPSI-7 or as SYSTEM-SHUTDOWN.

UPSI (User Program Status Indicator): Each UPSI is a User Program Status Indicator switch. At least one condition-name must be associated with each UPSI switch specified. UPSI-0 through UPSI-7 are COBOL names that identify program switches defined outside the COBOL program at execution time. Their contents are considered to be alphanumeric. A value of zero is off; a value of one is on. For the external setting of UPSI switches, see *Execution* in Chapter 9.

Each switch represents a bit from the 8-bit indicator-settings parameter of the OCL SWITCH statement as follows:

UPSI-0	First bit (leftmost)
UPSI-1	Second bit
UPSI-2	Third bit
.	.
.	.
.	.
UPSI-7	Eighth bit (rightmost)

One condition-name must be associated with each function-name-2; a second condition-name is optional. One condition-name can be associated with the ON status; another can be associated with the OFF status. Establishing condition-names for the ON or OFF status of a switch permits testing the setting of that switch.

Each condition-name is formed according to the rules for a user-defined word, and the condition-name must contain at least one alphabetic character.

Coding Example

```
A   B                                                    COBOL STATEM
    12   16   20   24   28   32   36   40   44
SPECIAL-NAMES.
    ATTRIBUTE-DATA IS WS-ATTRIBUTES,
    C01 IS NEXT-PAGE,
    CURRENCY-SIGN IS 'Y',
    IBM-ASCII IS STANDARD-1,
    LOCAL-DATA IS LOCAL-DATA-AREA,
    REQUESTOR IS WORKSTN,
    SYSTEM-CONSOLE IS SYSTM,
    SYSTEM-SHUTDOWN IS SHUTDOWN-SWITCH,
        ON STATUS SHUTDOWN-OCCURRED
    UPSI-0 IS UPSI-SWITCH-0,
        ON STATUS IS U0-ON,
        OFF STATUS IS U0-OFF,
    UPSI-1 IS UPSI-SWITCH-1,
        ON STATUS IS U1-ON,
        OFF STATUS IS U1-OFF.
```

The above coding example assigns mnemonic-names to the most commonly used function-names in the SPECIAL-NAMES paragraph.

In the Procedure Division, the UPSI switch status is tested through the associated condition-name(s). Each condition-name is the equivalent of a level-88 item. The associated mnemonic-name, if specified, is considered the conditional variable and can be used for qualification.

UPSI switches are tested by an IF statement. The current UPSI values are retrieved from the system and the test is performed against the switched settings. In an SRT program, current UPSI values are always associated with the requestor. In an MRT program, the current UPSI values are associated with whichever requestor has successfully executed the most recent READ operation.

UPSI switches are updated by a SET statement. The current UPSI values are retrieved from the system and these switch settings are updated as specified by the SET. The new switch settings are then returned to the system for future references.

Programming Notes: UPSI switches are useful for processing special conditions within a program, such as year-beginning or year-ending processing. At the beginning of the Procedure Division, an UPSI switch can be tested; if it is ON, the special branch is taken.

SYSTEM-SHUTDOWN: SYSTEM-SHUTDOWN is an internal switch that is set to the ON status when the system operator requests the STOP SYSTEM operation. This switch can then be tested by the program to terminate the job. The associated ON or OFF condition-names can be referenced in any conditional expression.

Alphabet-Name Clause

The alphabet-name clause provides a means of relating an alphabet-name to a specified character code set or collating sequence.

The alphabet-name specifies a collating sequence in one of the following:

- The PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph
- The COLLATING SEQUENCE option of the SORT or MERGE statement

If NATIVE is specified or the alphabet-name clause is not written, collating is done using the EBCDIC collating sequence.

If STANDARD-1 is specified, collating is done as if the data were translated from EBCDIC into ASCII. For more information on translating EBCDIC into ASCII, see Appendix G.

Literal Option: The literal option of the alphabet-name clause processes internal data in collating sequences other than NATIVE or STANDARD-1.

When the literal option is specified, the collating sequence to be used is specified by the user according to the following rules:

- The order in which literals appear specifies the ordinal number, in ascending sequence, of the character(s) in this collating sequence.
- Each numeric literal specified must be an unsigned integer and must have a value from 1 through 256 (the maximum number of characters in the EBCDIC character set). The value of each literal specifies the relative position of a character within the EBCDIC character set. For example, the literal 112 represents the EBCDIC character ?, the literal 234 represents the EBCDIC character Z, the literal 241 represents the EBCDIC numeric character 0. For more information on which numbers correspond to which letters, see Appendix G. Note, however, that the numbers in Appendix G begin at 0 and run to 255. You must add 1 to the number shown in Appendix G when you use the literal option.
- Each character in a nonnumeric literal represents that character in the EBCDIC set. If the nonnumeric literal contains more than one character, each character, starting with the leftmost, is assigned a successively ascending position within this collating sequence.
- Any EBCDIC characters not explicitly specified assume positions in this collating sequence higher than any of the explicitly specified characters. The relative order of the unspecified characters within the EBCDIC set remains unchanged.
- Within one alphabet-name clause, a given character must not be specified more than once.
- Each nonnumeric literal associated with a THROUGH or ALSO option must be one character in length.

- When the THROUGH option is specified, the contiguous EBCDIC characters beginning with the character specified by literal-1 and ending with the character specified by literal-2 are assigned successively ascending positions in this collating sequence. This sequence may be either ascending or descending within the original EBCDIC sequence. For example, if the characters Z through A are specified left to right, then for this collating sequence the ascending values, left to right, for the capital letters are: ZYXWVUTSRQPONMLKJIHGFEDCBA.

- When the ALSO option is specified, the EBCDIC characters specified as literal-1, literal-3, literal-4, and so on are assigned to the same position in this collating sequence. For example, if 'D' ALSO 'N' ALSO 112 ALSO '%' is specified, then for this collating sequence the characters D, N, ?, and % are all considered to be in the same position in the collating sequence.

- The character having the highest ordinal position in this collating sequence is associated with the figurative constant HIGH-VALUE. If more than one character has the highest position because the ALSO option is specified, the last character specified is considered to be the HIGH-VALUE character for procedural statements such as DISPLAY, or as the sending field in a MOVE statement. If the ALSO option example given above were specified as the high-order characters of the collating sequence, then the HIGH-VALUE character would be %.

- The character having the lowest ordinal position in this collating sequence is associated with the figurative constant LOW-VALUE. If more than one character has the lowest position because the ALSO option is specified, the first character specified is the LOW-VALUE character. If the ALSO option example given above were specified as the low-order characters of the collating sequence, then the LOW-VALUE character would be D.

Alphabet-Name Clause Examples: The following examples illustrate some uses for the alphabet-name clause.

If PROGRAM COLLATING SEQUENCE IS USER-SEQUENCE; if the alphabet-name clause is specified as USER-SEQUENCE IS 'DEF'; and if two Data Division items are defined as follows:

```
77 ITEM-1 PIC X(3) VALUE 'ABC'.
77 ITEM-2 PIC X(3) VALUE 'DEF'.
```

then the comparison IF ITEM-1 > ITEM-2 is true.

Characters D, E, and F are in ordinal positions 1, 2, and 3 of this collating sequence. Characters A, B, and C are in ordinal positions 197, 198, and 199 of this collating sequence.

If the alphabet-name clause is USER-SEQUENCE IS 1 THRU 247, 251 THRU 256, '7', ALSO '8', ALSO '9'; if all 256 EBCDIC characters have been specified; and if the two Data Division items are specified as follows:

```
77 ITEM-1 PIC X(3) VALUE HIGH-VALUE.
77 ITEM-2 PIC X(3) VALUE '787'.
```

then both of the following comparisons are true:

```
IF ITEM-1 = ITEM-2 . . .
IF ITEM-2 = HIGH-VALUE . . .
```

They compare as true because the values '7', '8', and '9' all occupy the same position (HIGH-VALUE) in this USER-SEQUENCE collating sequence.

If the alphabet-name clause is specified as USER-SEQUENCE IS 'E', 'D', 'F' and a table in the Data Division is defined as follows:

```
05 TABLE-A OCCURS 6 ASCENDING KEY IS
     KEY-A INDEXED BY INX-A.
     10 FIELD-A . . .
     10 KEY-A . . .
```

and if the contents in ascending sequence of each occurrence of KEY-A are A, B, C, D, E, G, then the results of the execution of a SEARCH ALL statement for this table will be invalid because the contents of KEY-A are not in ascending order. The proper ascending order would be E, D, A, B, C, G.

CURRENCY SIGN Clause

The literal that appears in the CURRENCY SIGN clause defines the currency symbol to be used in the PICTURE clause. The literal must be a one-character nonnumeric literal and must not be any of the following characters:

- Digits 0 through 9
- Alphabetic characters A B C D L P R S V X Z or the space
- Special characters * + - , . ; ( ) " / =

When the CURRENCY SIGN clause is omitted, only the dollar sign ($) may be used as the PICTURE symbol for the currency sign.

DECIMAL POINT IS COMMA Clause

When the DECIMAL POINT IS COMMA clause is specified, the functions of the period and the comma are exchanged in PICTURE character-strings and in numeric literals.

INPUT-OUTPUT SECTION

The Input-Output Section defines each file, identifies its external storage medium, assigns the file to one or more input/output devices, and also specifies information needed for efficient transmission of data between the external medium and the COBOL program.

The Input-Output Section is divided into two paragraphs: the FILE-CONTROL paragraph, which names and associates the files with the external media, and the I-O-CONTROL paragraph, which defines special input/output techniques to be used.

Format

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

The exact contents of the Input-Output Section depend on the file organization and access methods used to process the file. The following summary gives some background for the file processing techniques available in System/34 COBOL.

File Processing Summary

The method used to process a file in a COBOL program depends on the data organization of the file and on the access mode used.

Data organization is the permanent logical structure of the file, established at the time the file is created. Four types of data organization are available with System/34 COBOL: sequential, indexed, relative, and transaction. Relative data organization is also known as direct data organization on System/34.

An access mode is the manner in which records in a file are to be processed. Three access modes are available: sequential, random, and dynamic.

Disk File Processing

Figure 3-2 shows the allowable combinations of program-specified organization and access modes with disk file organization. Appendix H summarizes which clauses and statements are required and which clauses and statements are optional for each access method and device.

Program-Specified Organization	Program-Specified Access Modes	System/34 Disk File Organization		
		Sequential	Indexed	Direct
Sequential	Sequential	X	X[1]	X
Indexed	Sequential	–	X	–
	Random	–	X	–
	Dynamic	–	X	–
Relative	Sequential	X[1]	X[1]	X
	Random	X[1]	X[1]	X
	Dynamic	X[1]	X[1]	X

[1]The file must be opened as input.

Figure 3-2. Allowable Combinations of Program-Specified Organization and Access Modes With System/34 Disk File Organization

IBM Extension:

TRANSACTION File Processing

The TRANSACTION processing facilities of System/34 COBOL allow the user to read from and write to System/34 display stations. These facilities also permit communications with another application program on the same System/34 or on another system. COBOL TRANSACTION processing allows the user to define a file that supports display stations and SSP-ICF sessions in any combination.

TRANSACTION file processing of display stations interfaces with the formatted output of SFGR (Screen Format Generation Routine) and with WSDM (Work Station Data Management). These interfaces provide the ability to support single or multiple requestor programs (SRT/MRT), never ending programs (NEP), and acquired terminals. For further information and programming considerations, refer to the System Support Reference Manual and Chapter 7.

TRANSACTION file processing of SSP-ICF sessions provides access to the System/34 support for interactive communications. The Interactive Communications Feature supports program-to-program communication in the same or different systems. This support also includes remote job initiation. For information about how the Interactive Communications Feature works, refer to the Interactive Communications Feature Reference Manual.

The following paragraphs describe both the types of data organization, and the access modes available.

Data Organization for Disk Files

In a COBOL program, data organization for disk files can be sequential, indexed, or relative. Records can be fixed or variable in length. Variable length records are stored on the disk as fixed length records of the maximum size specified for the file.

Sequential Organization: With this organization, records are placed in the file consecutively, without keys, in the physical order established at file creation time. Once established, this relationship does not change, with the exception that a file can be extended.

Indexed Organization: With this organization, each record in the file has one embedded key that is associated with an index. The index provides a logical path to the data records according to the contents of the associated embedded record key data item.

When records are inserted, updated, or deleted, they are identified solely by the value of their record key. Thus, the value in each record key data item must be unique and must not be changed when the record is updated. The key used for any specific input/output request is known as the key of reference.

Relative Organization: With this organization, each record in the file is identified by its relative record number. The file can be thought of as a serial string of areas, each of which may contain one logical record. Each of these areas is identified by a relative record number; record storage and retrieval are based on this number. For example, the first record area is addressed by relative record number 1, and the 10th record area is addressed by relative record number 10 whether or not records have been written in the second through ninth record areas.

IBM Extension:

Data Organization for TRANSACTION Files

In a COBOL program, TRANSACTION files have TRANSACTION organization. With this organization, each record is associated with a session ID, which may be a display station session or with an SSP-ICF session. TRANSACTION organization signifies interactive input and output records that are not stored on an external medium.

Access Modes

Three access modes are available in COBOL: sequential, random, and dynamic.

Sequential Access Mode: This access method allows records of a file to be read and written in a serial manner. The order of reference is implicitly determined by the position of a record in the file.

Random Access Mode: This access method allows records to be read and written in a programmer-specified manner. The control of successive references to the file is expressed by specifically defined keys supplied by the user.

Dynamic Access Mode: This access method allows a specific input/output request to determine the access mode. Thus records may be processed sequentially and/or randomly.

Access Mode Allowed for Each File Type

Sequential Files: Files with sequential organization can be accessed only sequentially. The sequence in which records are accessed is the order in which the records were originally written.

Indexed Files: All three access modes are allowed.

In the sequential access mode, the sequence in which records are accessed is the ascending order of the record key value.

In the random access mode, the sequence in which records are accessed is controlled by the programmer. The desired record is accessed by placing the value of its record key in the RECORD KEY data item defined for that file.

In the dynamic access mode, the programmer may change at will from sequential access to random access by using appropriate input/output statements.

Relative Files: All three access modes are allowed.

In the sequential access mode, the sequence in which records are accessed is the ascending order of the relative record numbers of all records that currently exist within the file.

In the random access mode, the sequence in which records are accessed is controlled by the programmer. The desired record is accessed by placing its relative record number in a RELATIVE KEY data item. The RELATIVE KEY must not be defined within the record description entry for this file.

In the dynamic access mode, the programmer may change at will from sequential access to random access by using appropriate input/output statements.

IBM Extension:

TRANSACTION Files: Files with TRANSACTION organization can be accessed only sequentially. The sequence in which records are accessed is the order in which the transactions are received by the system.

FILE-CONTROL Paragraph

The FILE-CONTROL paragraph contains one or more file-control entries. A file-control entry associates a file in the COBOL program with an external medium, and this entry allows specification of file organization, access mode, and other information. The format of a file-control entry varies with the type of file described. The formats for the FILE-CONTROL paragraph are as follows:

Format 1–Sequential File Entries

```
SELECT [OPTIONAL] file-name

    ASSIGN TO assignment-name-1 [,assignment-name-2] . . .

    [ RESERVE integer-1 [AREA / AREAS] ]

    [ORGANIZATION IS SEQUENTIAL]

    [ACCESS MODE IS SEQUENTIAL]

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

Format 2–Indexed File Entries

```
SELECT file-name

    ASSIGN TO assignment-name-1 [,assignment-name-2] . . .

    [ RESERVE integer-1 [AREA / AREAS] ]

    ORGANIZATION IS INDEXED

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

    RECORD KEY IS data-name-2

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

Format 3–Relative File Entries

```
SELECT file-name

    ASSIGN TO assignment-name-1 [,assignment-name-2] . . .

    [ RESERVE integer-1 [AREA / AREAS] ]

    ORGANIZATION IS RELATIVE

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

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

Format 4–Sort or Merge File Entry

```
SELECT file-name ASSIGN TO assignment-name-1 [,assignment-name-2] . . .
```

When file-name specifies a sort or merge file, only the ASSIGN clause may follow the SELECT clause in this file-control entry. The ASSIGN clause associates the sort or merge file with a storage medium.

Format 5–TRANSACTION File Entry

```
SELECT file-name

    ASSIGN TO assignment-name

    ORGANIZATION IS TRANSACTION

    [ FILE STATUS IS data-name-1 [ , data-name-4] ]

    [ACCESS MODE IS SEQUENTIAL ]

    [CONTROL-AREA IS data-name-5 ].
```

FILE-CONTROL Paragraph—General Considerations

Each file described in an FD or SD entry in the Data Division must be described in only one entry in the FILE-CONTROL paragraph. Each file specified in a file-control entry must have a file description in the Data Division.

The key word FILE-CONTROL may appear only once, at the beginning of the FILE-CONTROL paragraph. The word FILE-CONTROL must begin in Area A, and it must be followed by a period and a space.

Each file-control entry must begin in Area B with a SELECT clause. The order in which other clauses appear is not significant.

Each clause within a file-control entry can optionally be separated from the next by a comma or semicolon followed by a space. Each file-control entry ends with a period and a space.

Each data-name must appear in a Data Division data description entry. Each data-name may be qualified but may not be subscripted or indexed.

IBM Extension:

TRANSACTION File Considerations: The TRANSACTION file must be named by a file control entry in the FILE-CONTROL paragraph. This entry also specifies other information related to the file. There may be only one TRANSACTION file in a program.

SELECT Clause

Each file-name specified in a SELECT clause must have an FD or SD entry in the Data Division. A file-name must conform to the rules for a COBOL user-defined name, must contain at least one alphabetic character, and must be unique within this program.

Sequential File Considerations: The OPTIONAL phrase can be specified only for input files with sequential organization. It must be specified for input files that are not necessarily present each time the object program is executed.

ASSIGN Clause (Printer and Disk Files)

The ASSIGN clause associates a file with an external medium. The assignment-name makes the association between the file and the external medium. Assignment-name has the following general stucture:

Device Type-Name.

The valid entries for each field of the assignment-name vary with the device. The format and valid value for each field are shown in Figure 3-3. The second and subsequent assignment-names are treated as comments.

Device Type:	PRINTER	Printer files
	DISK	Disk files
Name:	1- to 8-character field specifying the external name by which the file is known to the system. This is the name that appears in the NAME field on the OCL FILE or PRINTER statement.	

Note: When COBOL programs use the SORT or MERGE verbs, a user file cannot have WORK as an external name. The IBM Sort Utility uses a file by that name for storing intermediate results.

Figure 3-3. Assignment-Name Field Values for Printer and Disk Files

IBM Extension:

ASSIGN Clause (TRANSACTION Files)

The ASSIGN clause associates the TRANSACTION file with display stations and/or SSP-ICF sessions through the use of the assignment-name.
Assignment-name has the following structure:

```
Type [-Name          ]
     [-Name-Formats  ]
```

The values for each field of the assignment-name are as follows:

Type: WORKSTATION

Name: 1- to 8-character name that specifies the external name of the display screen format load member generated by SFGR that contains the screen formats. This field is not required if the file is to be used with SSP-ICF sessions only.

Formats: A two-digit numeric value that is equal to or greater than the number of formats in the SFGR load member referenced in the name field. The maximum value and the default value for the number of formats is 32.

The COBOL compiler constructs an internal table to hold information about each format. The value specified for formats determines how many 16-byte entries are in the internal table. The actual number of formats in your program does not influence the number of entries in this internal table.

RESERVE Clause

The RESERVE clause allows the user to modify the number of input/output areas (buffers) allocated by the compiler. This clause specifies that the number of buffers represented by integer be reserved for a disk file that is accessed sequentially. Print files use only 1 buffer.

The integer must have a value of 1 or 2. A minimum of one buffer is required for a file. If this clause is omitted, a minimum of one buffer area is reserved.

ORGANIZATION Clause

The ORGANIZATION clause specifies the logical structure of the file. The file organization is established at the time the file is created and cannot subsequently be changed. When the ORGANIZATION clause is omitted, ORGANIZATION IS SEQUENTIAL is assumed.

Sequential File Considerations: When ORGANIZATION IS SEQUENTIAL is specified or implied, a predecessor-successor relationship of the records in the file is established by the order in which records are placed in the file when it is created or extended.

Indexed File Considerations: When ORGANIZATION IS INDEXED is specified, the position of each logical record in the file is determined by the index created with the file and maintained by the system. The index is based on an embedded key within the file's records.

Relative File Considerations: When ORGANIZATION IS RELATIVE is specified, the position of each logical record in the file is determined by its relative record number.

IBM Extension:

TRANSACTION File Considerations: TRANSACTION organization signifies user-controlled input and output of records.

ACCESS MODE Clause

The ACCESS MODE clause defines the manner in which the records of the file are made available for processing. When this clause is omitted, ACCESS IS SEQUENTIAL is assumed.

Sequential File Considerations: For files with sequential organization, records in the file are accessed in the sequence established when the file is created or extended. Whether ACCESS IS SEQUENTIAL is specified or omitted, sequential access is always assumed.

Indexed File Considerations: For files with indexed organization, the access mode can be SEQUENTIAL, RANDOM, or DYNAMIC.

When ACCESS IS SEQUENTIAL is specified or implied, records in the file are accessed in the sequence of ascending record key values within the index.

When ACCESS IS RANDOM is specified, the value placed in the record key data item specifies the record to be accessed.

When ACCESS IS DYNAMIC is specified, records in the file can be accessed sequentially or randomly, depending on the form of the specific input/output request.

Relative File Considerations: For files with relative organization, the access mode can be SEQUENTIAL, RANDOM, or DYNAMIC.

When ACCESS IS SEQUENTIAL is specified or implied, records in the file are accessed in the ascending sequence of relative record numbers of existing records in the file.

When ACCESS IS RANDOM is specified, the value placed in the RELATIVE KEY data item specifies the record to be accessed.

When ACCESS IS DYNAMIC is specified, records in the file can be accessed sequentially or randomly, depending on the form of the specific input/output request.

The RELATIVE KEY phrase specifies the relative record number for a specific logical record in a relative file.

Data-name-3 is the RELATIVE KEY data item. It must be defined as an unsigned integer data item from 1 to 7 bytes in length and must not be defined in a record description entry associated with this relative file. That is, the RELATIVE KEY is not part of the record.

When ACCESS IS SEQUENTIAL is specified, the RELATIVE KEY phrase need not be specified unless the START statement is used. When the START statement is issued, the system uses the contents of the RELATIVE KEY data item to determine the record at which sequential processing is to begin.

If a value is placed in the RELATIVE KEY data item and a START statement is not issued, the value is ignored and processing begins with the first record in the file.

When ACCESS IS RANDOM or ACCESS IS DYNAMIC is specified, the RELATIVE KEY phrase must be specified. For each random processing request, the contents of the RELATIVE KEY data item are used to communicate a relative record number to the system.

IBM Extension:

TRANSACTION File Considerations: For files with TRANSACTION organization, the only allowable access mode is SEQUENTIAL.

RECORD KEY Clause (Indexed File)

The RECORD KEY clause must be specified for an indexed file. The RECORD KEY clause specifies the data item within the record that is the record key for an indexed file. The values contained in the record key data item must be unique among records in the file.

Data-name-2 is the RECORD KEY data item. It must be described as a fixed-length alphanumeric item within a record description entry associated with the file. The data description of data-name-2 and its relative location within the record must be the same as the ones used when the file was defined. Data-name-2 may be any fixed-length item within the record. It must be 29 bytes or less in length.

When two or more record descriptions are associated with a file, a similar field must appear in each description, and must be in the same relative position from the beginning of the record, although the same data-name-2 need not be used for both fields.

Programming Notes: No position in the record key should contain a hex FF (HIGH-VALUE) if the record is to be retrieved randomly by the indexed key or by the START verb.

FILE STATUS Clause

The FILE STATUS clause allows the user to monitor the execution of each input/output request for the file.

Data-name-1 is the status key data item. Data-name-1 must be defined in the Data Division as a two-character alphanumeric item and must not be defined in the File Section.

When the FILE STATUS clause is specified, the system moves a value into the status key data item after each input/output request that explicitly or implicitly refers to this file. The value indicates the execution status of the statement.

IBM Extension:

TRANSACTION File Considerations: Data-name-4 is the extended file status key data item and may be specified only for a TRANSACTION file. Data-name-4 must be defined in the Data Division as a four-character alphanumeric item and must not be defined in the File Section.

The extended file status key is used for major and minor return codes for SSP-ICF.

CONTROL-AREA Clause (TRANSACTION Files)

The CONTROL-AREA clause specifies the data item which receives feed-back information upon completion of a TRANSACTION file input/output operation. The information is in the fixed format as shown in Figure 3-4. Each item in the feed-back area is described as follows:

Bytes 1-2 (Function Key): The function key is a two-digit number inserted by the display station interface that identifies which function key the operator pressed to initiate the transaction. This item is updated only for a READ operation. The codes are as follows:

00	Enter key
01-24	COMMAND keys 1 through 24
90	ROLL UP Key
91	ROLL DOWN key
99	Undefined

Bytes 3-4 (Terminal ID): The symbolic identification of a display station or SSP-ICF session. This item is updated for every input/output operation.

Bytes 5-12 (Reserved): Reserved for future use.

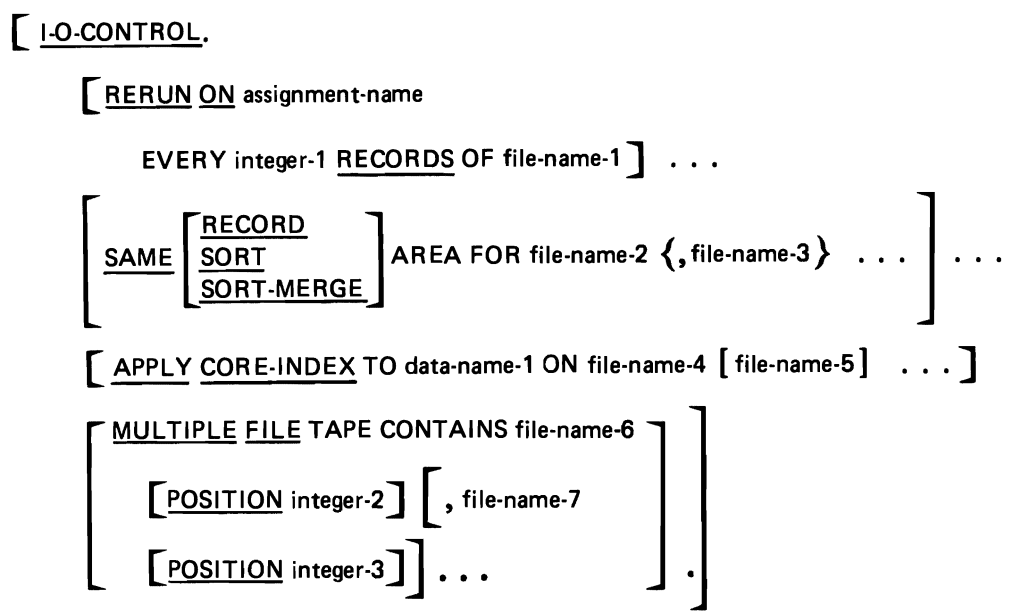
```
FILE-CONTROL.
      SELECT SCREEN-FILE ASSIGN TO
       WORKSTATION-MYFMTS-12
         ORGANIZATION IS TRANSACTION
         CONTROL-AREA IS TRANSACTION-CONTROL-AREA.
              .
              .
              .
WORKING-STORAGE SECTION.
01    TRANSACTION-CONTROL-AREA.
          03   FUNCTION-KEY      PIC 99.
          03   TERMINAL-ID       PIC X(2).
          03   RESERVED          PIC X(8).
```

Figure 3-4. Sample FILE-CONTROL Paragraph and CONTROL-AREA

I-O-CONTROL Paragraph

The I-O-CONTROL paragraph specifies when checkpoints are to be taken and what storage areas are to be shared by different files and optimization techniques. The I-O-CONTROL paragraph is optional in a COBOL program.

Format



The key word I-O-CONTROL can appear only once, at the beginning of the I-O-CONTROL paragraph. The word I-O-CONTROL must begin in Area A, and it must be followed by a period followed by a space.

Each clause within the I-O-CONTROL entry can optionally be separated from the next by a comma or semicolon followed by a space. The I-O-CONTROL entry ends with a period followed by a space.

IBM Extension:

The clauses in the I-O-CONTROL paragraph may appear in any order.

RERUN Clause

The RERUN clause specifies that checkpoint records are to be taken. A checkpoint record is a recording of the status of a user program and main storage resources at desired intervals. The contents of main storage are recorded on fixed disk at the time of the checkpoint and can be used to restart the program from that point.

Checkpoint records are written for the first record and every subsequent integer-1 number of records of file-name-1 that are processed. Checkpoint records are written on fixed disk in the file specified by assignment-name. This area contains two alternating checkpoint records, with each checkpoint record overlaying the oldest checkpoint taken.

Note: All files used in the program must be opened before the first checkpoint can be taken.

File-name-1 represents the file for which checkpoint records are to be written. It must be described with an FD or SD file description entry in the Data Division. The value of integer must not exceed 32767.

Note: File-name-1 cannot refer to a TRANSACTION file.

An assignment-name specified in a RERUN clause may not duplicate any assignment-name specified in an ASSIGN clause. The format of the assignment-name for the RERUN clause is:

DISK-file-name

where file-name is the name of the checkpoint record file for this job step. This file must be a new file. (The file-name must not be the label of a file already existing on the system even if the creation date is different.)

The user must not specify an OCL FILE statement when providing disk space for checkpoint records, as this file is allocated by the System/34 SSP.

Multiple RERUN clauses can be specified in a program as long as each file-name-1 is unique. The same assignment-name should be used for all RERUN statements in a program.

Checkpoint/restart is an optional facility of the SSP and must be specified during system configuration.

SAME Clause

The SAME clause specifies that two or more files are to use the same main storage area during processing. The files named in a SAME clause need not have the same organization or access.

The following discussion describes only the SAME RECORD AREA and SAME AREA options. The SAME SORT AREA and SAME SORT-MERGE AREA options are discussed under *SORT-MERGE* in Chapter 6.

The SAME RECORD AREA clause specifies that two or more files are to use the same main storage area for processing the current logical record. All the files may be open at the same time. A logical record in the shared storage area is considered to be both a logical record of each opened output file in this SAME RECORD AREA clause, and a logical record of the most recently read input file in this SAME RECORD AREA clause.

For physical sequential files, the area being shared includes all storage areas assigned to the files if the SAME AREA clause does not contain the RECORD option; therefore, it is not valid to have more than one of the files open at one time.

More than one SAME clause may be included in a program; however, the following restrictions apply:

- A specific file-name must not appear in more than one SAME AREA clause.
- A specific file-name must not appear in more than one SAME RECORD AREA clause.
- If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in that SAME AREA clause must appear in that SAME RECORD AREA clause. However, that SAME RECORD AREA clause may contain additional file-names that do not appear in that SAME AREA clause.
- The SAME AREA clause may have only one open file at a time.

Programming Notes: (1) Specification of the SAME AREA clause saves space in the object program. Note, however, the restrictions when this clause is used. (2) Specification of the SAME RECORD AREA clause does not necessarily save space in the object program. This clause allows transfer of data from one file to another with minimal data manipulation because the input/output areas of named files are identical, and all are available to the user.

APPLY Clause

IBM Extension: The APPLY clause may be specified only for an indexed file and is used to specify that the index is to be processed in main storage.

The APPLY clause is designed to increase system performance for randomly and dynamically accessed indexed files by allowing the user to specify that a main storage resident index is to be built for the files. Using a main storage resident master index reduces the time required to locate a record. This can improve performance on the START and READ instructions.

Data-name-1 specifies an area in main storage in which an index for file-name-4 and, if specified, file-name-5, . . . is to be built when the file is opened. Data-name must be a WORKING-STORAGE item and can be from 3 to 9999 bytes in length. It may not be an alphanumeric edited item.

Data-name-1 should not be referred to in Procedure Division statements when any file associated with it is open.

The same file-name can only be used once in each clause and cannot appear in more than one APPLY clause. If multiple file-names are specified for a given core index, only one file may be opened at any given time. When the REDEFINES clause is also specified, the two data-names that refer to the same storage area must not be associated with two files that are open at the same time.

Calculation of CORE-INDEX: For efficient processing, the CORE-INDEX specified should be large enough to contain one entry for each track in the index plus one delimiter entry for the CORE-INDEX. The *minimum* number of bytes required for a CORE-INDEX entry can be obtained by using the following formula:

*(key length + 3) * (number of tracks + 1)*

For example, if file INDEXT has one track that contains index entries and a key length of 4, the most efficient storage index is 14 bytes, because:

*(4 + 3) * (1 + 1) = 14*

If the file index occupies more than one track, use the following calculations to determine the amount of storage to set aside for index entries:

1. Use the CATALOG procedure to find the total number of records that the file can contain.

2. Determine the entry length by adding 3 to the key length.

3. Determine the number of keys in each sector by dividing 256 by the entry length. Drop the remainder.

4. Determine the number of sectors in the index by dividing the number of records in the file (the result of step 1) by the number of keys in each sector (the result of step 3). Round up the result.

5. Determine the number of tracks in the index by dividing the number of sectors (the result of step 4) by 60 if the disk has 27.1 megabytes or less. If the disk has 63.9 megabytes or more, divide by 64. If the quotient is not a whole number, round it up to the next whole number.

   To determine the number of megabytes on disk, the STATUS command can be used to display the session status. The AVAILABLE DISK SIZE field on the third session status screen shows the number of megabytes. (For information on the Status Session display, see the *Operator's Guide*.)

   If a precise number of tracks is not needed, the number of sectors can be divided by 60. Using 60 sectors per track allows data management enough room to construct one entry per track and ensures that data management will have enough spaces to create the storage index.

6. Add 1 to the result of step 5. Multiply this sum by the result of step 2. This is the number of bytes to allocate for the CORE-INDEX.

MULTIPLE FILE Clause

The MULTIPLE FILE clause is treated as comments.

Chapter 4. Data Division

DATA DIVISION CONCEPTS

The Data Division of a COBOL source program describes all the data to be processed by the object program. Two types of data can be processed: external data and internal data.

External Data

External data is contained in files. A file is a collection of data records existing on some input/output device. A file can be thought of as a group of physical records; it can also be thought of as a group of logical records. The Data Division source statements describe the relationship between physical and logical records.

A physical record is a unit of data that is treated as an entity when it is moved into or out of auxiliary storage. The size of a physical record is determined by the particular input/output device on which it is stored. The size does not necessarily have a direct relationship to the size or content of the logical information contained in the file.

A logical record is a unit of data whose subdivisions have a logical relationship. A logical record can itself be a physical record (that is, be contained completely in one physical unit of data), several logical records can be contained within one physical record, or one logical record can extend across several physical records.

Record description entries, which follow the FD entry for a specific file, describe the logical records in the file. These entries also describe the category and format of data within each field of the logical record and different values the data might be assigned.

The FD entry specifies the physical aspects of the data such as the size relationship between physical and logical records, the size and name(s) of the logical record(s), and labeling information.

Once the relationship between physical and logical records has been established, only logical records are made available to the COBOL program. Thus, in this manual, a reference to records means logical records unless the term physical records is used.

Internal Data

Program logic may develop additional data within storage. Such data is called internal data.

The concept of logical records applies to internal data as well as to external data. Internal data can thus be grouped into logical records and be defined by a series of record description entries. Items that need not be so grouped can be defined in independent data description entries.

Data Relationships

The relationships of all data to be used in a program are defined in the Data Division through a system of level indicators and level-numbers.

A level indicator, together with its descriptive entry, identifies each file description in a program. Level indicators are the highest level of any data hierarchy with which they are associated.

A level-number, together with its descriptive entry, indicates the properties of specific data. Level-numbers can be used to describe a data hierarchy. They can indicate that this data has a special purpose, and while they can be associated with and be subordinate to level indicators, they can also be used independently to describe internal data or data common to two or more programs.

DATA DIVISION ORGANIZATION

The Data Division is divided into three sections: the File Section, the Working-Storage Section, and the Linkage Section. Each section has a specific logical function within a COBOL source program, and each can be omitted from the source program when that logical function is not needed.

Format

```
DATA DIVISION.
[ FILE SECTION.
    [file-description-entry or sort-merge-
        file-description-entry] . . .
    {record-description-entry} . . .]
[ WORKING-STORAGE SECTION.
    [data-item-description-entry] . . .
    [record-description-entry] . . .]
[ LINKAGE SECTION.
    [data-item-description-entry] . . .
    [record-description-entry] . . .]
```

In the source program, the Data Division sections must appear in the order shown.

File Section

The File Section contains a description of all externally stored data (FD) and a description of each sort-merge file (SD) used in the program.

The File Section must begin with the header FILE SECTION followed by a period. The File Section contains file description entries and sort-merge file description entries. Each entry is followed by its associated record description entry (or entries).

In a COBOL program, the file description entries (beginning with the level indicators FD and SD) represent the highest level of organization in the File Section. The file description entry provides information about the physical structure and identification of a file, and gives the record-name(s) associated with that file. For further description of the format and the clauses required in a file description entry, see *File Description Entry* in this chapter. For a complete discussion of the sort-merge file description entry, see *Data Division—Sort/Merge* in Chapter 6.

The record description entry consists of a set of data description entries that describe the records contained within a particular file. More than one record description entry may be specified; each is an alternative description of the same storage area. For the format and the clauses required within the record description entry, see *Data Description* in this chapter.

Data areas described in the File Section should not be considered available for processing unless the file containing the data area is open.

Working-Storage Section

The Working-Storage Section can contain description records that are not part of data files but are developed and processed internally. These records are used for report description, counters, and other functions necessary in processing data.

The Working-Storage Section must begin with the section header WORKING-STORAGE SECTION followed by a period. The Working-Storage Section contains record description entries and data description entries for noncontiguous data items.

Data elements in the Working-Storage Section that bear a definite hierarchical relationship to one another must be grouped into records structured by level number.

Noncontiguous items in this section that bear no hierarchical relationship to one another need not be grouped into records provided they do not need to be further subdivided. Instead, they are classified and defined as noncontiguous elementary items. Each is defined in a separate data description entry that begins with the special level number 77. The format of the data description entry is the same as the format for the record description entry.

Linkage Section

The Linkage Section describes data made available from another program.

Record description entries and data description entries in the Linkage Section provide names and descriptions, but storage within the program is not reserved because the data area exists elsewhere. Any data description clause may be used to describe items in the Linkage Section with one exception: the VALUE clause may not be specified for any items other than level-88 items. For additional information, see *Subprogram Linkage* in Chapter 6.

FILE DESCRIPTION ENTRY

In a COBOL program, the file description entry (FD entry) or the sort-merge file description entry (SD entry) is the highest level of organization in the File Section.

Format 1–Sequential, Indexed, Relative Files

```
[ FILE SECTION.

[ FD file-name

    [ BLOCK CONTAINS [integer-1 TO ] integer-2 { RECORDS    } ]
                                               { CHARACTERS }
    [ RECORD CONTAINS [integer-3 TO ] integer-4 CHARACTERS ]

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

    [ VALUE OF implementor-name-1 IS { data-name-1 }
                                     { literal-1   }

        [ ,implementor-name-2 IS { data-name-2 } ] ... ]
                                 { literal-2   }

    [ DATA { RECORD IS  } data-name-3 [ ,data-name-4 ] ... ]
           { RECORDS ARE}

    [ LINAGE IS { data-name-5 } LINES [ ,WITH FOOTING AT { data-name-6 } ]
                { integer-5   }                          { integer-6   }

        [ ,LINES AT TOP { data-name-7 } ] [ ,LINES AT BOTTOM { data-name-8 } ] ]
                        { integer-7   }                      { integer-8   }

    [ CODE-SET IS alphabet-name ] .

  { record-description-entry } ... ] ... ]
```

Format 2–TRANSACTION File

```
FD file-name

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

LABEL { RECORDS ARE } OMITTED
      { RECORD IS   }

[ DATA { RECORD IS  } data-name-3 [ ,data-name-4 ] . . . ] .
       { RECORDS ARE }

{ record-description-entry } . . .
```

Coding Example

```
SEQUENCE  CONT.  A   B
(PAGE) SERIAL
1   3  4   6  7  8  12  16  20  24  28
003 010  DATA DIVISION.
    020  FILE SECTION.
    030  FD   FILE-NAME
    040       RECORD ~~~~~~~~~~
    050       LABEL RECORD ~~~~~
    060       DATA RECORD IS ~~~~.
    070  01   DESCRIPTION ~~~~~~.
    080       .
    090       .
    100       .
    110  WORKING-STORAGE SECTION.
    120  77   NAME-DESCRIPTION.
    130  01   RECORD-DESCRIPTION.
```

The above coding example specifies the most commonly used clauses for a format 1 file description entry.

The following example shows the Data Division in a program.

```
DATA DIVISION.
FILE SECTION.
FD  INPUT-DATA
    BLOCK CONTAINS 1 RECORDS
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORDS ARE GEN-INFO SALES-DATA.
01  GEN-INFO.
        03 EMPLOYEE-NAME.
              05 FIRST-NAME                          PIC X(12).
              05 LAST-NAME                           PIC X(12).
        03 SOC-SEC-NUMBER PIC 9(9).
        03 CHECK-SSN  REDEFINES SOC-SEC-NUMBER       PIC X(9).
        03 AGE                                       PIC 99.
        03 BIRTH-DATE.
              05 B-MONTH                             PIC 99.
              05 B-DAY                               PIC 99.
              05 B-YEAR                              PIC 99.
        03 ANNUAL-SALARY                             PIC 9(5)V99.
        03 CHECK-SALARY REDEFINES ANNUAL-SALARY      PIC X(7).
*   THIS REDEFINES WILL BE USED TO SEE IF THE FIELD IS BLANK.
        03 RECORD-ID                                 PIC X.
        03 FILLER                                    PIC X(31).
01  SALES-DATA.
        03 SALES-SSN                                 PIC 9(9).
        03 SALES-LOCATION                            PIC XX.
             88 MICHIGAN VALUE IS 'MI'.
             88 EASTERN-REGION VALUES ARE 'PA' 'NY'.
             88 HEADQUARTERS VALUES ARE 'BA' THRU 'BZ'.
        03 TOTAL-COMMISSION                          PIC 9(5)V99.
        03 RECORD-CODE                               PIC X.
        03 FILLER                                    PIC X(61).
FD  RECORD-OUT
    LABEL RECORDS ARE OMITTED
    RECORD CONTAINS 132 CHARACTERS
    LINAGE IS 60 LINES
    WITH FOOTING 59
    LINES AT TOP 3
    LINES AT BOTTOM 3
    DATA RECORD IS PRINT-OUT
01  PRINT-OUT PIC X(132).
WORKING-STORAGE SECTION.
77  RECORDS-IN PIC 9(6) VALUE ZEROS.
77  DECLARATIVE-ERRORS PIC 9(4) VALUE ZEROS.
77  EOF-SW PIC X VALUE ZERO.
77  BAD-DATA-COUNTER PIC 9(3) VALUE ZERO.
77  CHECK-IT PIC XX.
01  PRINT-FIELDS-EDITED.
        03 FILLER PIC X(14) VALUE SPACES.
        03 TOTAL-SALARY PIC $$$,$$$.$9BB.
        03 COMMISSION-COSTS PIC $**,***,***.99B.
        03 FILLER PIC X(65) VALUE ALL '-'.
        03 FILLER PIC X(12) VALUE '---END---JOB'.
01  SALARY-COUNTER PIC 9(6)V99 VALUE ZEROS.
01  COMMISSION-COUNTER PIC 9(6)V99 VALUE ZEROS.
```

The file description entry must begin with the level indicator FD followed by a space.

The clauses that follow file-name are optional in many cases; the order of their appearance is not significant.

However, at least one record description entry must follow the FD entry. When more than one record description entry is specified, each entry implies a redefinition of the same storage area. The last clause in the FD entry must be immediately followed by a period and a space.

IBM Extension:

Format 2-TRANSACTION File Considerations

A file description entry consists of a level indicator (FD), a file-name, and a series of independent clauses. For a TRANSACTION file, the independent clauses allowed are the RECORD CONTAINS clause, the LABEL RECORDS clause, and the DATA RECORDS clause. Only the LABEL RECORDS clause is required.

The LABEL RECORDS clause specifies whether or not labels are present. Label records must be omitted for a transaction file. This clause is required in every file description entry.

The RECORD CONTAINS clause and the DATA RECORDS clause are described under RECORD CONTAINS Clause and DATA RECORDS Clause, later in this chapter. The record definition must be large enough to hold the largest record defined by the SFGR formats or SSP-ICF records processed by the program.

File-Name

The file-name must follow the level indicator, and must be the same file-name as that specified in the SELECT clause of the associated file control entry.

The file-name must follow the rules of formation for a user-defined word; at least one character must be alphabetic. The file-name must be unique within this program.

BLOCK CONTAINS Clause

The BLOCK CONTAINS clause specifies the size of a physical record. When the BLOCK CONTAINS clause is omitted, the compiler assumes that records are not blocked. Thus, this clause can be omitted when each physical record contains only one complete logical record.

The BLOCK CONTAINS clause is used by the compiler to establish the input/output buffer size for a disk file. The BLOCK CONTAINS clause has no effect on the physical formatting of the file as it resides on disk. The size given or calculated for the BLOCK CONTAINS clause is rounded up by the compiler to the next higher multiple of 256, unless the size is a multiple of 256.

Format

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

Integer-1 and integer-2 must be nonzero unsigned integers.

When neither the CHARACTERS nor RECORDS option is specified, the CHARACTERS option is assumed.

RECORDS Option: When the RECORDS option is specified, the physical record size is expressed as the number of logical records contained in each physical record.

The compiler assumes that the block size must provide for integer-2 records of maximum size, and provides any additional space needed for control bytes.

Note: Maximum record size is 4096; maximum block size is 9999.

CHARACTERS Option: When the CHARACTERS option is specified or implied, the physical record size is specified as the number of character positions required to store the physical record no matter what USAGE the characters within the data record have.

If only integer-2 is specified, it specifies the exact character size of the physical record. When integer-1 and integer-2 are both specified, they represent, respectively, the minimum and maximum character size of the physical record.

The compiler assumes that the block size must provide for integer-2 characters even when integer-1 is provided.

RECORD CONTAINS Clause

The RECORD CONTAINS clause specifies the size of a file's data records.

Format

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

The RECORD CONTAINS clause is never required because the size of each record is completely defined in the record description entries. When this clause is specified, the following rules apply:

- Integer-3 and integer-4 must be unsigned, nonzero integers.
- When both integer-3 and integer-4 are specified, integer-3 specifies the size of the smallest data record, and integer-4 specifies the size of the largest data record.
- Integer-4 must not be specified alone unless all the records are the same size. If all records are the same size, integer-4 specifies the exact number of characters in the record.
- The record size must be specified as the number of character positions needed to store the record internally; that is, size is specified in terms of the number of bytes occupied internally by the record's characters, regardless of the number of characters used to represent the item within the record. The size of a record is determined according to the rules for obtaining the size of a group item. For a further description of record size, see the *USAGE Clause* in this chapter.

Note: When the RECORD CONTAINS clause is omitted, the record lengths are determined by the compiler from the record descriptions. When one of the entries within a record description contains an OCCURS DEPENDING ON clause, the compiler uses the maximum value of the variable length item to calculate the record length.

LABEL RECORDS Clause

The LABEL RECORDS clause specifies whether labels are present or omitted. The LABEL RECORDS clause is required in every FD entry.

Format

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

STANDARD Option: The STANDARD option specifies that labels conforming to system specifications exist for this file. This option must be specified for disk files.

OMITTED Option: The OMITTED option specifies that no labels exist for this file. This option must be specified for files assigned to unit record devices.

VALUE OF Clause

The VALUE OF clause serves only as documentation. It is used to specify the description of an item in the label records associated with this file.

Format

```
[ VALUE OF implementor-name-1 IS {data-name-1}
                                 {literal-1   }

    [ ,implementor-name-2 IS {data-name-2}
                             {literal-2   } ] ... ]
```

DATA RECORDS Clause

The DATA RECORDS clause serves only as documentation for the names of data records associated with this file. The DATA RECORDS clause is never required.

Format

```
[ DATA { RECORD IS  } data-name-3 [,data-name-4] . . . ]
       { RECORDS ARE}
```

The specification of more than one data-name indicates that this file contains more than one type of data record. Two or more record descriptions for this file occupy the same storage area. These records need not have the same description or length. The order in which the data-names are listed is not significant.

LINAGE Clause

The LINAGE clause specifies the depth of a logical page in terms of the number of lines. This clause also optionally specifies the line number at which the footing area begins, as well as the top and bottom margins of the logical page. There is not necessarily a relationship between the logical page size and the physical page size.

Format

```
[ LINAGE IS { data-name-5 } LINES [ , WITH FOOTING AT { data-name-6 } ]
            { integer-5 }                              { integer-6 }

    [ , LINES AT TOP { data-name-7 } ] [ , LINES AT BOTTOM { data-name-8 } ] ]
                     { integer-7 }                         { integer-8 }
```

The LINAGE clause may be specified only for printer files.

All integers must be unsigned. All data-names must be described as unsigned integer data items.

LINAGE Integer-5/Data-Name-5: Integer-5 or the value in data-name-5 specifies the number of lines that can be written and/or spaced on this logical page. The area of the page that these lines represent is called the page body. The value must be greater than zero.

WITH FOOTING Option: Integer-6 or the value in data-name-6 specifies the first line number of the footing area within the page body. The footing line number must be greater than zero, but it must not be greater than the number for the last line of the page body. The footing area extends between those two lines. If this option is not specified, the assumed value is equal to that of the page body (integer-5 or data-name-5).

LINES AT TOP Option: Integer-7 or the value in data-name-7 specifies the number of lines in the top margin of the logical page. If this option is not specified, zero is assumed.

LINES AT BOTTOM Option: Integer-8 or the value in data-name-8 specifies the number of lines in the bottom margin of the logical page. If this option is not specified, zero is assumed.

Figure 4-1 illustrates the use of each option of the LINAGE clause.

LINAGE Clause Considerations: The logical page size specified in the LINAGE clause is the sum of all values specified in each option except the FOOTING option. If the LINES AT TOP and/or the LINES AT BOTTOM options are zero, each logical page immediately follows the preceding logical page with no additional spacing provided.

At the time an OPEN OUTPUT statement is executed, the values of integer-5, integer-6, integer-7, and integer-8 are used to determine the page body, first footing line, top margin, and bottom margin of the logical page for this file. These values are then used for all logical pages printed for this file during a given execution of the program.

Data-name-5, data-name-6, data-name-7, and data-name-8 cause the following actions to take place:

- Their values at the time an OPEN OUTPUT is executed are used to determine the page body, the first footing line, the top margin, and the bottom margin for the first logical page only.
- Their values at the time a WRITE ADVANCING statement causes page ejection are used to determine the page body, first footing line, top margin, and bottom margin for the next succeeding logical page only.

LINAGE-COUNTER Special Register: For each FD entry containing a LINAGE clause, a separate LINAGE-COUNTER special register is generated. LINAGE-COUNTER is initialized to one when an OPEN statement for this file is executed. LINAGE-COUNTER is automatically modified by any WRITE statement for this file. When linage is specified, the linage counter is set at the top of the first page body.

When more than one LINAGE-COUNTER special register is referred to in the PROCEDURE DIVISION, the user must qualify each LINAGE-COUNTER with its related file-name. For example, LINAGE-COUNTER OF FILE-A.

The value in LINAGE-COUNTER at any given time is the line number at which the device is positioned within the current page. LINAGE-COUNTER may be referred to in Procedure Division statements; LINAGE-COUNTER must not be modified by these statements.

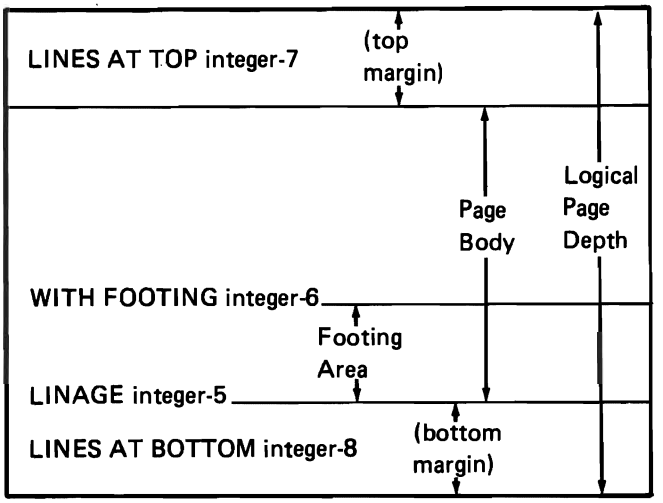


Figure 4-1. LINAGE Clause and Logical Page Depth

CODE-SET Clause

The CODE-SET clause is not required or used by the System/34 COBOL compiler. If it is inserted in the source program, the compiler treats this clause as a comment.

Format

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

DATA DESCRIPTION

All the data used in a COBOL program is described using a uniform system of representation. The basic concepts of data description are discussed in this chapter, as well as the actual COBOL clauses used to describe data.

Data Description Concepts

Most of the data processed by a COBOL program is presented in hierarchically arranged records. This is necessary because most data must be divided into subdivisions for processing. To subdivide such records, COBOL uses a hierarchical concept of levels.

For example, in a department store's customer file, one complete record could contain all data pertaining to one customer. Subdivisions within that record could be: customer name, customer address, account number, department number of sale, unit amount of sale, dollar amount of sale, previous balance, and other pertinent information.

Level Concepts

Because records must be divided into logical subdivisions, the concept of levels is inherent in the structure of a record. Once a record has been subdivided, it can be further subdivided to provide more detailed data references.

The basic subdivisions of a record (that is, those fields that are not further subdivided) are called elementary items. Thus, a record can be made up of a series of elementary items, or it may itself be an elementary item.

It may be necessary to refer to a set of elementary items. Thus, elementary items can be combined into group items. Groups can be combined into a more inclusive group that contains two or more subgroups. Thus, within one hierarchy of data items, an elementary item can belong to more than one group item.

Level-Numbers

A system of level-numbers specifies the organization of elementary and group items into records. Special level-numbers are also used to identify data items used for special purposes.

Each group and elementary item in a record requires a separate entry, and each must be assigned a level-number. The following level-numbers are used to structure records:

01:

This level-number specifies the record itself and is the most inclusive level-number possible. A level-01 entry may be either a group item or an elementary item.

02-49:

These level-numbers specify group and elementary items within a record. Less inclusive data items are assigned higher (not necessarily consecutive) level-numbers.

A group item includes all group and elementary items following it until a level-number less than or equal to the level-number of this group is encountered.

All elementary or group items immediately subordinate to one group item must be assigned identical level-numbers that are higher than the level-number of this group item.

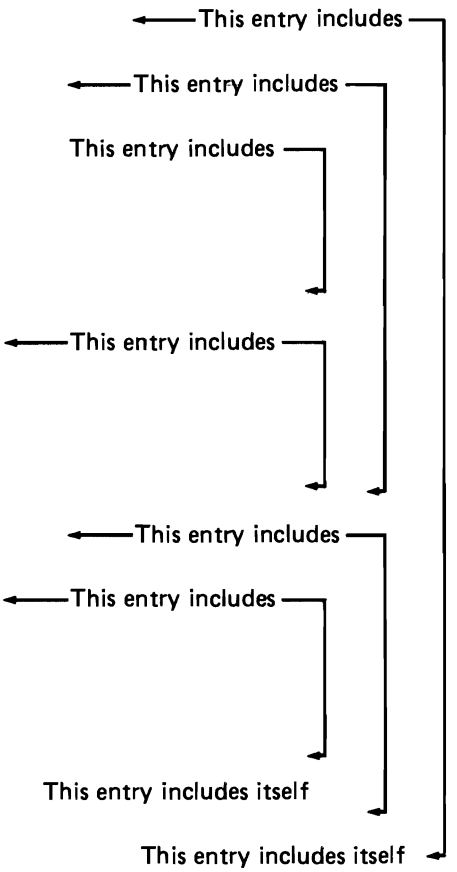
Figure 4-2 illustrates the level number concept. Notice that all groups immediately subordinate to the level-01 entry have the same level-number. Notice also that elementary items from different subgroups do not necessarily have the same level number, and that elementary items can be specified at any level within the hierarchy. Figure 4-2 shows the COBOL record-description entry in the left portion of the figure; it shows the subdivision of the entry in the right portion of the figure.

Note: Level-numbers 01 through 09 can also be written as 1 through 9.

The COBOL record description entry is written as follows:

```
01   RECORD-ENTRY.
     05   GROUP-1.
          10   SUBGROUP-1.
               15   ELEM-1   PIC . . . .
               15   ELEM-2   PIC . . . .
          10   SUBGROUP-2.
               15   ELEM-3   PIC . . . .
               15   ELEM-4   PIC . . . .
     05   GROUP-2.
          15   SUBGROUP-3.
               25   ELEM-5   PIC . . . .
               25   ELEM-6   PIC . . . .
          15   SUBGROUP-4   PIC . . . .
     05   GROUP-3 PIC . . . .
```

The items included in the hierarchy of each level are indicated below:



The storage arrangement is illustrated below:

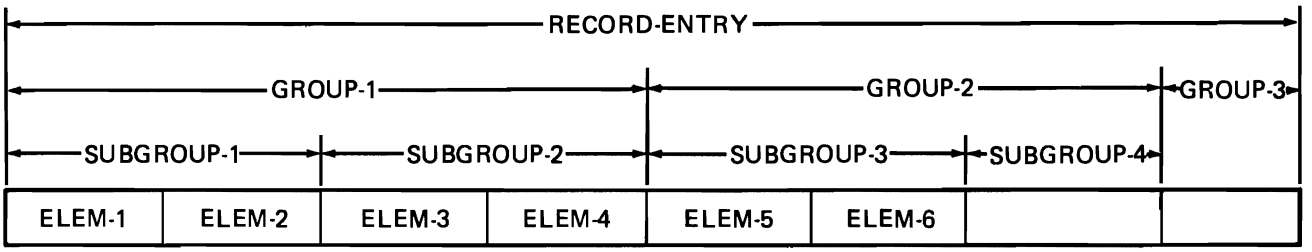


Figure 4-2. Storage Arrangement of Record Description Entry

Special Level-Numbers

Special level-numbers identify items that do not structure a record. The following are special level-numbers:

66:

This level number identifies elementary or group items described by a RENAMES clause. Such items regroup previously defined data items.

77:

This level number identifies independent data description entries in the Working-Storage or Linkage Section. These items are not subdivisions of other items, and are not themselves subdivided.

88:

This level number identifies any condition-name entry that is associated with a particular value of a conditional variable. An example is given under *VALUE Clause* in this chapter.

Note: Level-77 and level-01 entries in the Working-Storage Section and Linkage Section must be given unique data-names because neither can be qualified. If subordinate data-names can be qualified, they need not be unique.

Indentation

Successive data description entries may begin in the same column as preceding entries, or they may be indented according to level-number. Indentation is useful for documentation, but it does not affect the action of the compiler.

Classes of Data

All data used in a COBOL program can be divided into four classes and six categories. Every elementary item in a program belongs to one of the classes as well as one of the categories. Every group item belongs to the alphanumeric class even if the subordinate elementary items belong to another class and category. Figure 4-3 shows the relationship of data classes and categories.

Level of Item	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric edited Alphanumeric edited Alphanumeric
	Boolean	Boolean
Group	Alphanumeric	Alphabetic Numeric Numeric edited Alphanumeric edited Alphanumeric Boolean

Figure 4-3. Classes and Categories of Data.

IBM Extension:

Boolean Data Facilities

Boolean data provides a means of modifying and passing the values of the indicators associated with the display screen formats. A Boolean value of 0 is the indicator's off status while a Boolean value of 1 is the indicator's on status.

A Boolean literal contains a single 0 or 1 and is enclosed in quotes and immediately preceded by an identifying B. The Boolean literal is defined as either B'0' or B'1'. A Boolean character occupies one byte. The figurative constant ZERO can be used as a Boolean literal, and the reserved word ALL is valid with a Boolean literal. The Boolean ZERO is the fill character for Boolean data.

Standard Alignment Rules

The standard alignment rules for positioning data in an elementary item depend on the data category of the receiving item (that is, the item into which the data is placed).

Numeric Items: When a numeric item is the receiving item, the following rules apply:

- The data is aligned on the assumed decimal point and, if necessary, truncated or padded with zeros. (An assumed decimal point is one that has logical meaning but does not exist as a character in the data.)
- If a decimal point is not explicitly specified, the receiving item is treated as though an assumed decimal point is specified immediately to the right of the field. The data is then treated as in the preceding rule.

Numeric Edited Items: The data is aligned on the decimal point and, if necessary, truncated or padded with zeros at either end, except when editing causes replacement of leading zeros.

Alphanumeric, Alphanumeric Edited, Alphabetic: For these data categories, the following rules apply:

- The data is aligned at the leftmost character position and, if necessary, truncated or padded with spaces at the right.
- If the JUSTIFIED clause is specified for alphanumeric or alphabetic receiving items, the above rule is modified as described in the JUSTIFIED clause.

Note: The JUSTIFIED clause must not be specified for any item for which editing is specified.

Standard Data Format

COBOL makes data description as machine independent as possible. For this reason, the properties of the data are described in a standard data format rather than a machine-oriented format.

The standard data format uses the decimal system to represent numbers no matter what base is used by the system. The nonnumeric data can contain any characters that are in the native character set, that is, nonnumeric data is not limited to just the COBOL character set or the nonnumeric COBOL characters.

Character-String and Item Size

In COBOL, the size of an elementary item is determined through the number of character positions specified in its PICTURE character-string. In storage, however, the size is determined by the actual number of bytes the item occupies as determined by the combination of its PICTURE character-string and its USAGE clause.

Normally, when an arithmetic item is moved from a longer field to a shorter one, the compiler truncates the data to the number of characters represented in the shorter item's PICTURE character-string.

For example, if a sending field with PICTURE S99999 and containing the value +12345 is moved to a COMPUTATIONAL receiving field with PICTURE S99, the data is truncated to +45.

Signed Data

There are two categories of algebraic signs used in COBOL: operational signs and editing signs.

Operational Signs

Operational signs (+, -) are associated with signed numeric items and indicate their algebraic properties. The internal representation of an algebraic sign depends on the item's USAGE clause and optionally upon its SIGN clause. Zero is considered a unique value regardless of the operational sign. An unsigned field is always assumed to be positive or zero.

Editing Signs

Editing signs are associated with numeric edited items. Editing signs are PICTURE symbols (+, -, CR, DB) that identify the sign of the item in edited output.

DATA DESCRIPTION ENTRY

A record description entry or a data description entry specifies the characteristics of a particular data item. The maximum length for any item that is not otherwise restricted is 32767 bytes. The four general formats are:

Format 1

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

    [REDEFINES clause]
    [USAGE clause]
    [SIGN clause]
    [OCCURS clause]
    [SYNCHRONIZED clause]
    [JUSTIFIED clause]
    [BLANK WHEN ZERO clause]
    [VALUE clause]
    [PICTURE clause]
```

Format 2–RENAMES Clause

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

Format 3

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

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

Format 4–Boolean Data

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

    [REDEFINES clause]
    [USAGE clause]
    [OCCURS clause]
    [SYNCHRONIZED clause]
    [JUSTIFIED clause]
    [VALUE clause]
    [PICTURE clause]
    [INDICATOR clause]
```

Format 1

This format is used for record description entries in all sections and for level-77 entries in the Working-Storage and Linkage Sections. The following rules apply:

- Level-number can be any number from 01 through 49 or 77.
- The clauses can be written in any order, with two exceptions:
  - The data-name/FILLER clause must immediately follow the level-number.
  - When specified, the REDEFINES clause must immediately follow the data-name clause.
- The PICTURE clause must be specified for every elementary item except index data items.
- The BLANK WHEN ZERO, JUSTIFIED, PICTURE, and SYNCHRONIZED clauses are valid only for elementary items.
- Either a space, or a comma or a semicolon followed by a space, must separate clauses.
- Each record description entry must end with a period followed by a space.

Format 2–RENAMES Clause

The RENAMES clause specifies alternative, possibly overlapping, groupings of elementary data items. This clause allows a single data-name to rename a group of data items within a record.

One or more RENAMES entries can be written for a logical record. All RENAMES entries associated with one logical record must immediately follow that record's last data description entry. A level-66 entry cannot rename a level-01, level-77, level-88, another level-66 entry, or another data-name that contains an INDICATOR clause.

The compiler does not compensate for internal formats such as packed decimal, zoned decimal and binary that are renamed to a different format.

Note: You can use the RENAMES clause to rename an INDICATOR data item. However, the new data-name does not have an INDICATOR value associated with it and cannot be used as an indicator.

Data-name-1 identifies an alternative grouping of data items. It cannot be used as a qualifier; it can be qualified only by the names of level indicator entries or level-01 entries.

Note: Level-number 66 and data-name-1 are not part of the RENAMES clause itself, and are included in the format only for clarity.

Data-name-2 or data-name-3 identifies the original grouping of elementary data items; that is, they must name elementary or group items within the associated level-01 entry and must not be the same data-name. Both data-names may be qualified.

The OCCURS clause must not be specified in the data entries for data-name-2 and data-name-3, or for any group entry to which they are subordinate. In addition, the OCCURS DEPENDING ON clause must not be specified for any item occupying storage between data-name-2 and data-name-3.

Data-Name-2 Option: When data-name-3 is not specified, data-name-2 can be either a group item or an elementary item. When data-name-2 is a group item, data-name-1 is treated as a group item. When data-name-2 is an elementary item, data-name-1 is treated as an elementary item.

Data-Name-2 THRU Data-Name-3 Option: When data-name-3 is specified, data-name-1 is a group item that includes all elementary items:

- Starting with data-name-2 (if it is an elementary item) or the first elementary item within data-name-2 (if it is a group item)
- Ending with data-name-3 (if it is an elementary item) or the last elementary item within data-name-3 (if it is a group item).

The key words THRU and THROUGH are equivalent.

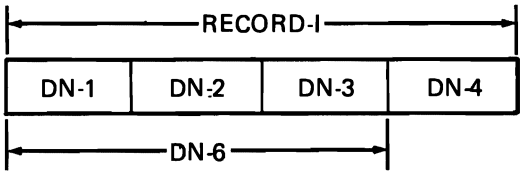
The leftmost character in data-name-3 must not precede that in data-name-2; the rightmost character in data-name-3 must follow that in data-name-2. This means that data-name-3 cannot be subordinate to data-name-2.

Valid and invalid specifications of the RENAMES clause are given in Figure 4-4.

COBOL Specifications | **Storage Layouts**

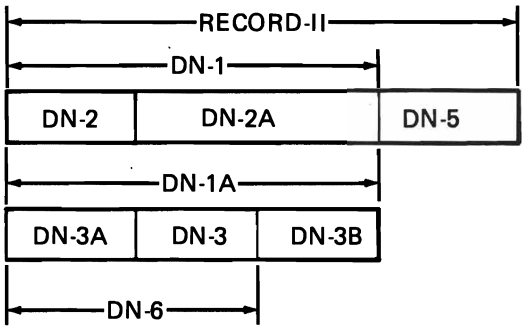
Example 1 (Valid)

```
01   RECORD-I.
     05  DN-1 . . .  .
     05  DN-2 . . .  .
     05  DN-3 . . .  .
     05  DN-4 . . .  .
66   DN-6 RENAMES DN-1 THROUGH DN-3.
```



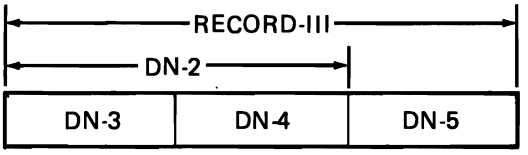
Example 2 (Valid)

```
01   RECORD-II.
     05  DN-1.
         10  DN-2 . . .  .
         10  DN-2A . . .  .
     05  DN-1A REDEFINES DN-1.
         10  DN-3A . . .  .
         10  DN-3 . . .  .
         10  DN-3B . . .  .
     05  DN-5 . . .  .
66   DN-6 RENAMES DN-2 THROUGH DN-3.
```



Example 3 (Invalid)

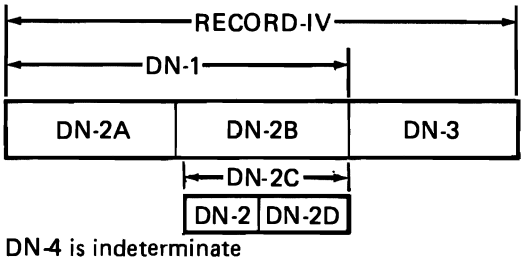
```
01   RECORD-III.
     05  DN-2.
         10  DN-3 . . .  .
         10  DN-4 . . .  .
     05  DN-5 . . .  .
66   DN-6 RENAMES DN-2 THROUGH DN-3.
```



DN-6 is indeterminate

Example 4 (Invalid)

```
01   RECORD-IV.
     05  DN-1.
         10  DN-2A . . .  .
         10  DN-2B . . .  .
         10  DN-2C REDEFINES DN-2B.
             15  DN-2 . . .  .
             15  DN-2D . . .  .
     05  Dn-3 . . .  .
66   DN-4 RENAMES DN-1 THROUGH DN-2.
```



DN-4 is indeterminate

Figure 4-4. Valid and Invalid Specifications of the RENAMES Clause

Format 3

This format describes condition-names. A condition-name is a user-specified name that associates value(s) and/or a range(s) of values with a conditional variable.

A conditional variable is a data item that can assume one or more values that can, in turn, be associated with a condition-name. The following rules for condition-name entries apply:

- Any entry beginning with level-number 88 is a condition-name entry.
- The condition-name entries associated with a particular conditional-variable must immediately follow the conditional variable entry. The conditional variable can be any elementary data description entry except another condition-name, index data item, or level-66 entry.
- A condition-name can be associated with a group item data description entry. The following rules apply:
  - The condition-name value must be specified as a nonnumeric literal or figurative constant.
  - The size of the condition-name value must not exceed the sum of the sizes of all the elementary items within the group.
  - No element within the group may contain a JUSTIFIED or SYNCHRONIZED clause.
  - No USAGE other than USAGE IS DISPLAY may be specified within the group.
- Condition-names can be specified both at the group level and at subordinate levels within the group.
- The relation test implied by the definition of a condition-name at the group level is performed in accordance with the rules for comparison of nonnumeric operands regardless of the nature of elementary items within the group.
- Either a space or a comma or a semicolon followed by a space, must separate successive operands.
- Each entry must end with a period followed by a space.
- The condition-name must not be qualified when used in a REDEFINES clause.

Examples of both elementary and group condition-name entries are given under *VALUE Clause* in this chapter.

IBM Extension:

Format 4–Boolean Data

This format is used for Boolean data items in all sections. The following rules apply:

- USAGE must be defined implicitly or explicitly as DISPLAY.
- In the OCCURS clause, the ASCENDING/DESCENDING key is not valid for Boolean data items.
- The INDICATOR clause must be specified at an elementary level only.
- A Boolean data item may be compared only with another Boolean data item.
- Only EQUAL or NOT EQUAL comparisons are allowed for Boolean data items.
- Boolean data items must be used for SFGR indicators with TRANSACTION files.

Level-Numbers

The level-number specifies the hierarchy of data within a record and also identifies special-purpose data entries.

Format

```
level-number
```

The following rules for level-numbers apply:

- A level-number begins a data description entry, a regrouped item, or a condition-name entry.
- Level-numbers 01 and 77 must begin in Area A.
- Level-numbers 02-49, 66, and 88 may begin in either Area A or Area B and must be followed by a space.
- Single-digit level-numbers 1 through 9 may be substituted for level-numbers 01 through 09.

Data-Name or FILLER Clause

A data-name explicitly identifies the data being described; the key word FILLER specifies an item that is never explicitly referenced in the program.

Format

```
data-name
FILLER
```

In a data description entry, either the data-name or the key word FILLER must be the first word following the level-number. The data-name identifies a data item by referring to the field, not to a particular value. This data item can assume a number of different values during the course of a program.

A data-name can begin anywhere in Area B. A data-name requires a period at the end of the entry, and it must contain at least one alphabetic character.

Entries at level-numbers 01 and 77 in the Working-Storage and Linkage Sections cannot be qualified, and therefore require unique data-names. Subordinate data-names that can be qualified do not require unique data-names.

The key word FILLER specifies an elementary item in a record that is never explicitly referred to. The word FILLER may be written anywhere in Area B. A period is required at the end of the entry.

In a MOVE CORRESPONDING statement, an ADD CORRESPONDING statement, or a SUBTRACT CORRESPONDING statement, FILLER items are ignored.

IBM Extension: A FILLER item can be used as a group item definition. Subordinate data items may then be referenced by the appropriate data-name.

REDEFINES Clause

The REDEFINES clause indicates that the same storage area can contain different data items. Redefinition can save storage by allowing the same area to be used for different purposes.

Format

```
level-number data-name-1 REDEFINES data-name-2
```

Level-number and data-name-1 are not part of the REDEFINES clause itself, and are included in the format only for clarity.

If specified, the REDEFINES clause must be the first entry following data-name-1.

The level-number of data-name-1 and data-name-2 must be identical and must not be level 66 or level 88.

Data-name-2 is the redefined item.

Data-name-1 is the redefining item and is an alternative description for the data-name-2 area.

Implicit redefinition is assumed when more than one level-01 entry subordinate to an FD entry is written. In such level-01 entries, the REDEFINES clause must not be specified.

Redefinition begins at data-name-1 and ends when a level-number less than or equal to that of data-name-2 is encountered. No entry having a level-number numerically lower than those of data-name-1 and data-name-2 may occur between these entries.

In the following example, A is the redefined item, and B is the redefining item. Redefinition begins with B and includes the two subordinate items B-1 and B-2. Redefinition ends when the level-05 item C is encountered.

```
05 A PICTURE X(6).
05 B REDEFINES A.
     10 B-1 PICTURE X(2).
     10 B-2 PICTURE 9(4).
05 C PICTURE 99V99.
```

The data description entry for data-name-2, the redefined item, cannot contain a REDEFINES clause or an OCCURS clause. However, the redefined item may itself be subordinate to an item that contains either clause. If the redefined item is subordinate to an OCCURS clause, data-name-2 in the REDEFINES clause (the redefined item) must not be subscripted or indexed.

The redefined item, the redefining item, and any items subordinate to them cannot contain an OCCURS DEPENDING ON clause.

When data-name-1, the redefining item, is specified with a level-number other than 01, it must specify a storage area of the same size as the redefined item data-name-2.

Multiple redefinitions of the same storage area are permitted. The entries giving the new descriptions of the storage area must immediately follow the description of the redefined area without intervening entries that define new character positions. Multiple redefinitions must all use the data-name of the original entry that defined this storage area. For example:

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

The redefining entry (identified by data-name-1) and any subordinate entries must not contain any VALUE clauses. This rule does not apply to condition-name entries.

Data items within an area can be redefined without their lengths being changed. For example:

```
05  NAME-2.
    10 SALARY PICTURE XXX.
    10 SO-SEC-NO PICTURE X(9).
    10 MONTH PICTURE XX.
05 NAME-1 REDEFINES NAME-2.
    10 WAGE PICTURE XXX.
    10 EMP-NO PICTURE X(9).
    10 YEAR PICTURE XX.
```

Data items can also be rearranged within an area. For example:

```
05 NAME-2.
    10 SALARY PICTURE XXX.
    10 SO-SEC-NO PICTURE X(9).
    10 MONTH PICTURE XX.
05 NAME-1 REDEFINES NAME-2.
    10 EMP-NO PICTURE X(6).
    10 WAGE PICTURE 999V999.
    10 YEAR PICTURE XX.
```

When an area is redefined, all descriptions of the area are always in effect; that is, redefinition does not cause any data to be erased and does not supersede the previous description. Thus, if B REDEFINES A has been specified, either of the two procedural statements MOVE X TO B and MOVE Y TO A could be executed at any point in the program.

In the first case, the area described as B would assume the value of X. In the second case, the same physical area (described now as A) would assume the value of Y. If the second statement is executed immediately after the first, the value of Y replaces the value of X in the one storage area.

The USAGE of a redefining data item need not be the same as that of a redefined item. This does not, however, cause any change in existing data. For example:

```
05 B PICTURE 99 USAGE DISPLAY VALUE 8.
05 C REDEFINES B PICTURE S99 USAGE
     COMPUTATIONAL-4.
05 A PICTURE S99 USAGE COMPUTATIONAL-4.
```

The bit configuration of the DISPLAY value 8 is 1111 0000 1111 1000. Redefining B does not change the bit configuration of the data in the storage area. Therefore, the two statements, ADD B TO A and ADD C TO A give different results. In the first case, the value 8 is added to A (because B has USAGE DISPLAY). In the second statement, the value -48 is added to A (because C has USAGE COMPUTATIONAL-4), and the bit configuration (truncated to 2 decimal digits) in the storage area has the binary value -48.

Unexpected results may occur when a redefining item is moved to a redefined item (that is, if B REDEFINES C and the statement MOVE B TO C is executed). Unexpected results may also occur when a redefined item is moved to a redefining item (from the previous example, unexpected results occur if the statement MOVE C TO B is executed).

The REDEFINES clause may be specified for an item within the scope of any area being redefined (that is, an item subordinate to a redefined item). For example:

```
05 REGULAR-EMPLOYEE.
   10 LOCATION PICTURE A(8).
   10 GRADE PICTURE X(4).
   10 SEMI-MONTHLY-PAY PICTURE
      9999V99.
   10 WEEKLY-PAY REDEFINES
      SEMI-MONTHLY-PAY
      PICTURE 999V999.

05 TEMPORARY-EMPLOYEE REDEFINES
   REGULAR-EMPLOYEE.
   10 LOCATION PICTURE A(8).
   10 FILLER PICTURE X(6).
   10 HOURLY-PAY PICTURE 99V99.
```

The REDEFINES clause may also be specified for an item subordinate to a redefining item. For example:

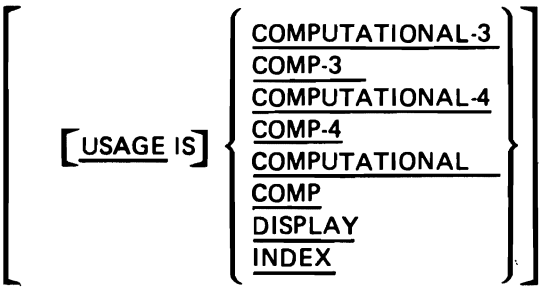
```
05 REGULAR-EMPLOYEE.
   10 LOCATION PICTURE A(8).
   10 GRADE PICTURE X(4).
   10 SEMI-MONTHLY-PAY
      PICTURE 999V999.

05 TEMPORARY-EMPLOYEE REDEFINES
   REGULAR-EMPLOYEE.
   10 LOCATION PICTURE A(8).
   10 FILLER PICTURE X(6).
   10 HOURLY-PAY PICTURE 99V99.
   10 CODE-H REDEFINES HOURLY-PAY
      PICTURE 9999.
```

USAGE Clause

The USAGE clause specifies the format of a data item in storage. The USAGE clause can be specified for an entry at any level. However, if it is specfied at the group level, it applies to each elementary item in the group. The usage of an elementary item cannot contradict the usage of a group to which the elementary item belongs.

The USAGE clause specifies the format in which data is represented in storage. The format may be restricted if certain Procedure Division statements are used.

Format



When the USAGE clause is not specified at either the group or elementary level, USAGE IS DISPLAY is assumed.

INDEX Option: The USAGE IS INDEX clause specifies that the data item named has an indexed format and, therefore, is an index data item. The index data item is an elementary item that can be used to save index-name values for future reference.

The USAGE IS INDEX clause is described in detail under *Table Handling* in Chapter 6.

DISPLAY Option: The DISPLAY option can be explicit or implicit. It specifies that the data item is stored in character form, one character per eight-bit byte. This corresponds to the form in which information is represented for keyboard input or for printed output. USAGE IS DISPLAY is valid for the following types of items:

- Alphabetic
- Alphanumeric
- Alphanumeric edited
- Numeric edited
- Zoned decimal (numeric)
- Boolean

Alphabetic, alphanumeric, alphanumeric edited, Boolean, and numeric edited items are discussed in the description of the PICTURE clause later in this chapter.

Zoned Decimal Items: These items are sometimes referred to as external decimal items. Each digit of a number is represented by a single byte. The four high-order bits of each byte are zone bits; the four high-order bits of the low-order byte represent the sign of the item. If the number is positive, these four bits contain a hexadecimal F. If the number is negative, these four bits contain a hexadecimal D. The four low-order bits of each byte contain the value of the digit. When zoned decimal items are used for computations, the compiler performs the necessary conversions. The maximum length of a zoned decimal item is 18 digits.

The PICTURE character-string of a zoned item may contain only 9s, the operational sign symbol S, the assumed decimal point V, and one or more Ps.

Examples of zoned decimal items are shown in Figure 4-4.

Computational Options: The term computational refers to the following options of the USAGE clause:

COMPUTATIONAL or COMP (zoned decimal)

IBM Extension: COMPUTATIONAL-3 or COMP-3 (packed decimal)

COMPUTATIONAL-4 or COMP-4 (binary)

A computational item represents a value to be used in arithmetic operations and must be numeric. If the USAGE of a group item is described with any of these options, it is the elementary items within the group that have this usage. The group itself is considered nonnumeric and cannot be used in numeric operations, except with the CORRESPONDING option. The maximum length of a computational item is 18 decimal digits.

The PICTURE of a computational item may contain only:

9 (one or more numeric character positions)

S (one operational sign)

V (one implied decimal point)

P (one or more decimal scaling positions)

The COMPUTATIONAL option is in zoned decimal format. Each digit of the number is represented by a single byte. The four leftmost bits of each byte are zone bits; the four leftmost bits of the rightmost byte represent the sign of the item. The four rightmost bits of each byte contain the value of the digit.

IBM Extension: A zoned decimal item may contain any of the digits 0 through 9, plus a sign.

The COMPUTATIONAL-3 option is specified for packed decimal items. Such an item appears in storage as two digits per byte, with the sign contained in the four rightmost bits of the rightmost byte. If the number is positive, these four bits contain a hexadecimal F. If the number is negative, these four bits contain a hexadecimal D.

A packed decimal item may contain any of the digits 0 through 9 plus a sign. If the PICTURE of a packed decimal item does not contain an S, the sign position is occupied by a bit configuration that is interpreted as positive, but does not represent an overpunch.

The COMPUTATIONAL-4 option is specified for binary data items. Such items have decimal equivalents consisting of the decimal digits 0 through 9, plus a sign.

The amount of storage occupied by a binary data item depends on the number of decimal digits defined in its PICTURE clause:

Digits in PICTURE Clause	Storage Occupied
1 through 4	2 bytes
5 through 9	4 bytes
10 through 18	8 bytes

The leftmost bit of the storage area is the operational sign.

Examples of packed decimal and binary items are shown in Figure 4-5.

Item	Description	Value	Internal Representation*
Zoned Decimal	PIC S9999 DISPLAY	+1234	F1 F2 F3 F4
		-1234	F1 F2 F3 D4
		1234	F1 F2 F3 F4
	PIC 9999 DISPLAY	+1234	F1 F2 F3 F4
		-1234	F1 F2 F3 F4
		1234	F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN LEADING	+1234	F1 F2 F3 F4
		-1234	D1 F2 F3 F4
		1234	F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN TRAILING SEPARATE	+1234	F1 F2 F3 F4 4E
		-1234	F1 F2 F3 F4 60
		1234	F1 F2 F3 F4 4E
	PIC S9999 DISPLAY SIGN LEADING SEPARATE	+1234	4E F1 F2 F3 F4
		-1234	60 F1 F2 F3 F4
		1234	4E F1 F2 F3 F4
Packed Decimal	PIC S9999 COMP-3	+1234	01 23 4F
		-1234	01 23 4D
	PIC 9999 COMP-3	+1234	01 23 4F
		-1234	01 23 4F
Binary	PIC S9999 COMP-4	+1234	04 D2
		-1234	FB 2E
	PIC 9999 COMP-4	+1234	04 D2
		-1234	04 D2

*The internal representation of each byte is shown as two hex digits. The bit configuration for each digit is as follows:

Hex Digit	Bit Configuration	Hex Digit	Bit Configuration
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Notes:

1. The leftmost bit of a binary number represents the sign: 0 is positive, 1 is negative.
2. Negative binary numbers are represented in twos complement form.
3. Hex 4E represents the EBCDIC character +. Hex 60 represents the EBCDIC character -.
4. Specification of SIGN TRAILING (without the SEPARATE CHARACTER option) is the equivalent of the standard action of the compiler.

Figure 4-5. Internal Representation of Numeric Items

SIGN Clause

The SIGN clause specifies the position and mode of representation of the operational sign for a numeric entry.

Format

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

The SIGN clause may be specified only for a signed numeric data description entry (that is, one whose PICTURE character-string contains an S), or for a group item that contains at least one such elementary entry. USAGE IS DISPLAY must be specified either explicitly or implicitly.

Only one SIGN clause may apply to any one data description entry. The SIGN clause is required only when an explicit description of the properties and/or position of the operational sign is necessary.

The SIGN clause defines the position and mode of representation of the operational sign for the numeric data description entry to which it applies, or for each signed numeric data description entry subordinate to the group to which it applies.

If the SEPARATE CHARACTER option is not specified, then:

- The operational sign is presumed to be associated with the LEADING or TRAILING digit position (whichever is specified) of the elementary numeric data item.
- The character S in the PICTURE character-string is not counted in determining the size of the item (in terms of standard data format characters).

If the SEPARATE CHARACTER option is specified, then:

- The operational sign is presumed to be the LEADING or TRAILING character position (whichever is specified) of the elementary numeric data item. This character position is not a digit position.
- The character S in the PICTURE character string is counted in determining the size of the data item (in terms of standard data format characters).
- + is the character used for the positive operational sign.
- - is the character used for the negative operational sign.
- If one of the character + or - is not present in the data at object time, an error occurs, and the program terminates abnormally.

Every numeric data description entry whose PICTURE contains the symbol S is a signed numeric data description entry. If the SIGN clause is also specified for such an entry and conversion is necessary for computations or comparisons, the conversion takes place automatically.

If no SIGN clause is specified for a signed numeric data description entry, the position and mode of representation for the operational sign is determined as explained in the USAGE clause description.

OCCURS Clause

The OCCURS clause specifies tables whose elements can be referred to by indexing or subscripting. It is described under *Data Division – Table Handling* in Chapter 6.

IBM Extension:

OCCURS Clause with Boolean Data Items

If the OCCURS clause and the INDICATOR clause are both specified at an elementary level, a table of Boolean data items is defined with each element in the table corresponding to an external indicator.

INDICATOR Clause

The INDICATOR clause is used to associate an SFGR indicator number with a Boolean data item. The format is:

```
INDICATOR integer
```

Integer must be greater than or equal to 1, and less than or equal to 99.

The INDICATOR clause must be specified at an elementary level only.

Since an indicator can contain only a value of zero or one, it must be associated only with a Boolean data-item.

OCCURS Clause with the INDICATOR Clause:

If the OCCURS clause and the INDICATOR clause are both specified at an elementary level, a table of Boolean data items is defined with each element in the table corresponding to an external indicator. The first element in the table corresponds to the indicator number specified in the INDICATOR clause, the second element corresponds to the indicator which sequentially follows the indicator specified by the INDICATOR clause.

For example, if the following is coded:

```
07 SWITCHES PIC 1 OCCURS 10 TIMES
INDICATOR 16.
```

then:

SWITCHES (1) corresponds to SFGR indicator 16,
SWITCHES (2) corresponds to SFGR indicator 17, . . .
SWITCHES (10) corresponds to SFGR indicator 25.

SYNCHRONIZED Clause

The SYNCHRONIZED clause specifies the alignment of an elementary item on a proper boundary in storage.

Format

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

The SYNCHRONIZED clause is treated as documentation only. The SYNCHRONIZED clause is never required. It may appear only at the elementary level. SYNC is an abbreviation for SYNCHRONIZED and has the same meaning.

JUSTIFIED Clause

The JUSTIFIED clause overrides standard positioning rules for a receiving item of the alphabetic or alphanumeric categories.

Format

The JUSTIFIED clause may be specified only at the elementary level. JUST is an abbreviation for JUSTIFIED and has the same meaning.

The JUSTIFIED clause must not be specified for a numeric item or for any item for which editing is specified. The JUSTIFIED clause must not be specified with level-66 (RENAMES) or level-88 (condition-name) entries.

When the JUSTIFIED clause is specified for a receiving item, the data is aligned at the rightmost character position in the receiving item, and:

- If the sending item is larger than the receiving item, the leftmost characters are truncated.
- If the sending item is smaller than the receiving item, the unused character positions at the left are filled with spaces.

When the JUSTIFIED clause is omitted, the rules for standard alignment are followed.

The following shows the difference between standard and justified alignment:

Alignment	Sending Field Value	Receiving Field Value
Standard	THE	THEbb
Justified right	THE	bbTHE

BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause specifies that an item is to be filled entirely with spaces when its value is zero.

Format

The BLANK WHEN ZERO clause may be specified only for elementary numeric or numeric edited items. When it is specified for a numeric item, the item is considered to be a numeric edited item.

If the BLANK WHEN ZERO clause is specified, the item contains nothing but spaces when its value is zero.

The BLANK WHEN ZERO clause must not be specified for level-66 or level-88 items.

IBM Extension: When both the BLANK WHEN ZERO clause and the asterisk (*) as a suppression symbol are specified for the same data description entry, zero suppression editing overrides the function of the BLANK WHEN ZERO clause.

VALUE Clause

The VALUE clause specifies the initial contents of a data item, or the value(s) associated with a condition-name. The two formats for the VALUE clause are as follows:

Format 1

```
[ VALUE IS literal ]
```

Format 2

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

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

Level-number 88 and condition-name are not part of the Format 2 VALUE clause itself, and are included in the format only for clarity. The use of the VALUE clause differs with the Data Division section in which it is specified.

File and Linkage Sections: The VALUE clause must be used only in condition-name entries.

Working-Storage Section: The VALUE clause is used in condition-name entries. It is also used to specify the initial value of any data item; the item assumes the specified value at the beginning of program execution. If the initial value is not explicitly specified, it is unpredictable.

General Considerations

The key words THRU and THROUGH are equivalent.

The VALUE clause must not be specified for any item whose length is variable.

For group entries, the VALUE clause must not be specified if the entry or an entry subordinate to it contains any of the following clauses: JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE DISPLAY).

The VALUE clause must not conflict with other clauses in the data description entry or in the data description of this entry's hierarchy. The following rules apply:

- Wherever a literal is specified, a figurative constant may be substitued.
- If the item is numeric, all VALUE clause literals must be numeric literals. If the literal defines the value of a Working-Storage item, the literal is aligned according to the rules for numeric moves with one additional restriction: the literal must not have a value that requires truncation of nonzero digits. If the literal is signed, the associated PICTURE character-string must contain a sign symbol (S).
- All numeric literals in a VALUE clause of an item must have a value that is within the range of values indicated by the PICTURE clause for that item. For example, for PICTURE 99PPP, the literal must be within the range 1000 through 99000 or zero. For PICTURE PPP99, the literal must be within the range .00000 through .00099.

- If the item is an elementary or group alphabetic, alphanumeric, alphanumeric edited, or numeric edited item, all VALUE clause literals must be nonnumeric literals. The number of characters in the literal must not exceed the size of the item.

- The functions of the editing characters in a PICTURE clause are ignored in determining the initial appearance of the item described. However, editing characters are included in determining the size of the item. Therefore, any editing character must be included in the literal. For example, if the item is defined as PICTURE +999.99 and the value is to be +12.34, then the VALUE clause should be specified as VALUE '+012.34'.

- A maximum of 32,767 bytes may be initialized by means of a single VALUE clause.

Format 1 Considerations

This format specifies the initial value of a data item in storage. Initialization is independent of any BLANK WHEN ZERO or JUSTIFIED clause specified.

A Format 1 VALUE clause must not be specified for an entry that contains or is subordinate to an entry that contains a REDEFINES or OCCURS clause.

If the VALUE clause is specified at the group level, the literal must be a nonnumeric literal or a figurative constant. The group area is initialized without consideration for the subordinate entries within this group. In addition, the VALUE clause must not be specified for subordinate entries within this group.

IBM Extension:

Boolean Considerations: The allowable values for a Boolean literal are B'0', B'1', and ZERO/S.

Format 2 Considerations

This format associates a value, values, and/or range(s) of values with a condition-name. Each such condition-name requires a separate level-88 entry.

The VALUE clause is required in a condition-name entry and must be the only clause in the entry. Each condition-name entry is associated with a preceding conditional variable. Thus, every level-88 entry must always be preceded either by the entry for the conditional variable or by another level-88 entry when several condition-names apply to one conditional variable. Such level-88 entries implicitly have the PICTURE characteristics of the conditional variable.

Every condition-name can be qualified by the name of its associated conditional variable and by the qualifier(s) of the conditional variable. If the associated conditional variable requires subscripts or indexes, each procedural reference to the condition-name must be subscripted or indexed as required for the conditional variable.

When only literal-1 is specified, the condition-name is associated with a single value.

When literal-1, literal-3, and so on are specified, the condition-name is associated with several single values.

When literal-1 THRU literal-2 is specified, the condition-name is associated with one range of values.

When literal-1 THRU literal-2, literal-3 THRU literal-4, and so on are specified, the condition-name is associated with more than one range of values. Literal-1 must be less than literal-2, literal-3 less than literal-4, and so on.

One or more single values and one or more ranges of values may be specified in a single Format 2 VALUE clause.

The type of literal in a condition-name entry must be consistent with the data type of the conditional variable. In the following example, CITY-COUNTY-INFO, COUNTY-NO, and CITY are conditional variables; the associated condition-names immediately follow the level-number 88. The PICTURE associated with COUNTY-NO limits the condition-name value to a 2-digit numeric literal. The PICTURE associated with CITY limits the condition-name value to a 3-character nonnumeric literal. Any values for the condition-names associated with CITY-COUNTY-INFO cannot exceed 5 characters, and the literal (because this is a group item) must be nonnumeric:

```
05 CITY-COUNTY-INFO.
     88 BRONX              VALUE '03NYC'.
     88 BROOKLYN           VALUE '24NYC'.
     88 MANHATTAN          VALUE '31NYC'.
     88 QUEENS             VALUE '41NYC'.
     88 STATEN-ISLAND      VALUE '43NYC'.
   10 COUNTY-NO            PICTURE 99.
     88 DUTCHESS           VALUE 14.
     88 KINGS              VALUE 24.
     88 NEW YORK           VALUE 31.
     88 RICHMOND           VALUE 43.
   10 CITY                 PICTURE X(3).
     88 BUFFALO            VALUE 'BUF'.
     88 NEW-YORK-CITY      VALUE 'NYC'.
     88 POUGHKEEPSIE       VALUE 'POK'.
05 POPULATION. . .
```

The following example shows the use of the THRU option. In this example, the number of miles a person drives to work each day is categorized.

```
05 MILEAGE PIC 9(2)V9.
     88 LOW VALUE 0 THRU 09.9.
     88 MED VALUE 10.0 THRU 19.9.
     88 HIGH VALUE 20.0 THRU 99.9.
```

Condition-names are used procedurally in condition-name conditions, and are described under *Conditional Expressions* in Chapter 5.

PICTURE Clause

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

Format

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

The PICTURE clause must be specified for every elementary item except an indexed data item. The PICTURE clause may be specified only at the elementary level. PIC is an abbreviation for PICTURE and has the same meaning.

The character-string is made up of certain COBOL characters used as symbols. The allowable combinations determine the category of the data item. The character-string may contain a maximum of 30 characters.

Symbols Used in the PICTURE Clause

The functions of each PICTURE clause symbol are defined in the following list. Any punctuation character appearing within the PICTURE character-string is not considered a punctuation character, but rather as a PICTURE character-string symbol.

A Each A in the character-string represents a character position that can contain only a letter of the alphabet or a space.

B Each B in the character-string represents a character position into which the space character will be inserted.

P The P indicates an assumed decimal scaling position, and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character P is not counted in the size of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric edited items or in items that appear as arithmetic operands. In any operation converting data from one form of internal representation to another, if the item being converted is described with the PICTURE symbol P, each digit position described by a P is considered to contain the value zero, and the size of the item is considered to include these zero digit positions.

For example, PICTURE PPP99 DISPLAY defines a 2-character item whose value is zero or ranges from .00001 through .00099. PICTURE 99PPP DISPLAY defines a 2-character item whose value is zero or ranges from 1000 through 99000.

The scaling position character P can appear only to the left or right of the other characters in the string as a continuous string of Ps within a PICTURE description. The sign character S and the assumed decimal point V are the only characters which can appear to the left of a leftmost string of Ps. Because the scaling position character P implies an assumed decimal point (to the left of the Ps if the Ps are leftmost PICTURE characters; to the right of the Ps if the Ps are rightmost PICTURE characters), the assumed decimal point symbol V is redundant as either the leftmost or rightmost character within such a PICTURE description.

S The symbol S is used in a PICTURE character-string to indicate the presence (but not the representation or, necessarily, the position) of an operational sign. The sign must be written as the leftmost character in the PICTURE string. An operational sign indicates whether the value of an item involved in an operation is positive or negative. The symbol S is not counted in determining the size of the elementary item, unless an associated SIGN clause specifies the SEPARATE CHARACTER option.

V The V is used in a character-string to indicate the location of the assumed decimal point and can appear only once in a character-string. The V does not represent a character position and, therefore, is not counted in the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string, the V is redundant.

X Each X in the character-string represents a character position that may contain any allowable character from the EBCDIC set.

Z Each Z in the character-string represents a leading numeric character position. When that position contains a zero, the zero is replaced by a space character. Each Z is counted in the size of the item.

IBM Extension:

1 A single 1 indicates a Boolean data item. If a 1 appears in the PICTURE character-string, it must be the only character.

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

0 Each zero in the character-string represents a character position into which the numeral zero will be inserted. Each zero is counted in the size of the item.

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

, Each comma in the character-string represents a character position into which a comma will be inserted. This character is counted in the size of the item. The comma insertion character cannot be the last character in the PICTURE character-string.

. When a period appears in the character-string, it is an editing symbol that represents the decimal point for alignment purposes. In addition, it represents a character position into which a period will be inserted. This character is counted in the size of the item. The period insertion character cannot be the last character in the PICTURE character-string.

Note: For a given program, the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. In this exchange, the rules for the period apply to the comma, and the rules for the comma apply to the period wherever they appear in a PICTURE clause.

+, -, CR, DB

These symbols are used as editing sign control symbols. Each symbol represents the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in one character-string. Each character used in the symbol is counted in determining the size of the data item.

* Each asterisk (check protect symbol) in the character-string represents a leading numeric character position into which an asterisk will be placed when that position contains a zero. Each asterisk is counted in the size of the item.

IBM Extension: Within a given data description entry, the use of the check protect symbol overrides the BLANK WHEN ZERO clause.

'CS' The currency symbol in the character-string represents a character position into which a currency symbol is to be placed. The currency symbol in a character-string is represented either by the symbol $ or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph of the Environment Division. The currency symbol is counted in the size of the item.

Note: Because the currency symbol can be replaced in the CURRENCY SIGN clause, the term 'CS' is used throughout this book rather than the actual currency symbol ($).

Figure 4-6 gives the order in which PICTURE clause symbols must be specified.

First Symbol / Second Symbol		Non-Floating Insertion Symbols									Floating Insertion Symbols						Other Symbols						
		B	0	/	,	.	{+ −}[1]	{+ −}[1]	{CR DB}	$	{Z *}[1]	{Z *}[1]	{+ −}[1]	{+ −}[1]	$[2]	$[2]	9	A X	S	V	P[1]	P[1]	1[3]
Non-Floating Insertion Symbols	B	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x	
	0	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x	
	/	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x	
	,	x	x	x	x	x	x			x	x	x	x	x	x	x				x		x	
	.	x	x	x	x		x			x	x		x		x		x						
	{+ −}																						
	{+ −}	x	x	x	x	x				x	x	x			x	x	x			x	x	x	
	{CR DB}	x	x	x	x	x				x	x	x			x	x	x			x	x	x	
	$						x																
Floating Insertion Symbols	{Z *}	x	x	x	x		x			x	x												
	{Z *}	x	x	x	x	x	x			x	x	x								x		x	
	{+ −}	x	x	x	x					x			x										
	{+ −}	x	x	x	x	x				x			x	x						x		x	
	$	x	x	x	x		x								x								
	P	x	x	x	x	x	x								x	x				x		x	
Other Symbols	9	x	x	x	x	x	x			x	x		x		x		x	x	x	x		x	
	A X	x	x	x													x	x					
	S																						
	V	x	x	x	x		x			x	x		x		x		x		x		x		
	P	x	x	x	x		x			x	x		x		x		x		x		x		
	P						x			x									x	x		x	

[1] Non-floating insertion symbols + and −, floating insertion symbols Z, *, +, −, and $, and other symbol P appear twice in the above table. The leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position. The second appearance of the symbol in the table represents its use to the right of the decimal point position.

[2] $ is the default value for the currency symbol. This value may be replaced by a character specified in the currency SIGN clause. At least one of the symbols A, X, Z, 9, or *, or at least two of the symbols +, −, or $ must be present in a PICTURE string.

[3] The character 1 must appear alone in the character string.

An X at an intersection indicates that the symbol(s) at the top of the column may, in a given character-string, appear anywhere to the left of the symbol(s) at the left of the row.

Braces ({ }) indicate items that are mutually exclusive.

Figure 4-6. PICTURE Clause Symbol Order

Character-String Representation: The following symbols may appear more than once in one PICTURE character-string:

A B P X Z 9 0 / , + - * 'CS'

Each time one of these symbols appears in the character-string, it represents an occurrence of that character or set of allowable characters in the data item.

An integer enclosed in parentheses immediately following any of these symbols specifies the number of consecutive occurrences of that symbol. The number of consecutive occurences may not exceed 32767.

For example, the following two PICTURE clause specifications are equivalent:

PICTURE IS $99999.99CR

PICTURE IS $9(5).99CR

The following five symbols may each appear only once in one PICTURE character-string:

S V . CR DB

Data Categories and PICTURE Considerations: The allowable combinations of PICTURE symbols determine the data category of the item. Rules for each category follow.

Alphabetic items–the following rules apply:

- The PICTURE character-string can contain only the symbols A and B.
- The contents of the item in standard data format must consist of any of the 26 letters of the alphabet and the space character.
- USAGE DISPLAY must be either specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal.

Numeric items–the following rules apply:

- The PICTURE character-string can contain only the symbols 9, P, S, and V.
- The number of digit positions must range from 1 through 18.
- The contents of a numeric item must be a combination of the digits 0 through 9. The numeric item may contain an operational sign. If the PICTURE contains an S, the contents of the item are treated as a positive or negative value, depending on the operational sign present in the data. If the PICTURE does not contain an S, the contents of the item are treated as an absolute value.
- If a VALUE clause is specified for an elementary numeric item, the literal must be numeric. If a VALUE clause is specified for a group item consisting of elementary numeric items, the group is considered alphanumeric, and the literal must therefore be nonnumeric.
- The USAGE of the item can be DISPLAY or COMPUTATIONAL.

  IBM Extension: IBM implementation also allows the USAGE to be COMPUTATIONAL-3 or COMPUTATIONAL-4.

Examples of numeric items are shown in Figure 4-7.

PICTURE	Valid Range of Values
9999	0 through 9999
S99	-99 through +99
S999V9	-999.9 through +999.9
PPP999	0 through .000999
S999PPP	-1000 through -999000 and +1000 through +999000 or zero

Figure 4-7. Examples of Numeric Items

Alphanumeric items–the following rules apply:

- The PICTURE character-string must consist of either:
  - The symbol X entirely.
  - Combinations of the symbols A, X, and 9. The item is treated as if the character-string contained only the symbol X. A PICTURE character-string containing all A's or all 9's does not define an alphanumeric item.
- The contents of the item in standard data format may be any allowable characters from the EBCDIC character set.
- USAGE DISPLAY must be either specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal.

Alphanumeric edited items–the following rules apply:

- The PICTURE character-string can contain the symbols:

  A X 9 B 0 /

- The string must contain at least one of the following combinations:
  - At least one B and at least one X
  - At least one 0 and at least one X
  - At least one X and at least one /
  - At least one A and at least one 0
  - At least one A and at least one /
- The contents of the item in standard data format may be any allowable character from the EBCDIC character set.
- USAGE DISPLAY must be either specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal. The literal is treated exactly as specified; no editing is performed.
- Alphanumeric edited items are subject to only one type of editing–simple insertion using the symbols 0, B, and /.

Numeric edited items–the following rules apply:

- The PICTURE character-string can contain the following symbols:

  B P V Z 9 0 / , . + - CR DB * 'CS'

  The combinations of symbols allowed are determined from the PICTURE clause symbol order allowed (see Figure 4-4), and the editing rules (see the following section). The following additional rules also apply:
  - The string must contain at least one of the following symbols:

    B / Z 0 , . * + - CR DB

  - The number of digit positions represented in the character-string must be in the range of 1 through 18 inclusive.
  - The total number of character positions in the string (including editing characters) must not exceed 30.
- The contents of those character positions representing digits in standard data format must be one of the digits 0 through 9.
- USAGE DISPLAY must be either specified or implied.
- Any associated VALUE clause must specifiy a nonnumeric literal. The literal is treated exactly as specified; no editing is performed.

IBM Extension:

Boolean items–the following rule applies:

- The string must contain a single character 1.

PICTURE Clause Editing

There are two general methods of performing editing in a PICTURE clause: by insertion, or by suppression and replacement.

There are four types of insertion editing: simple insertion, special insertion, fixed insertion, and floating insertion. There are two types of suppression and replacement editing: zero suppression and replacement with asterisks and zero suppression and replacement with spaces.

The type of editing allowed for an item depends on its data category. The type of editing and the insertion symbols that are valid for each category are shown in Figure 4-8.

Category	Type of Editing	Valid Insertion Symbols
Alphabetic	Simple insertion	B
Numeric	None	None
Alphanumeric	None	None
Alphanumeric edited	Simple insertion	B 0 /
Numeric edited	All	B 0 / ,
Boolean	None	None

Figure 4-8. Valid Editing for Each Data Category

Simple Insertion Editing: This type of editing is valid for alphabetic, alphanumeric edited, and numeric edited items. The valid insertion symbols for each category are shown in Figure 4-8.

Each insertion symbol is counted in the size of the item, and represents the position within the item where the equivalent characters will be inserted. Examples of simple insertion editing are shown in Figure 4-9.

PICTURE	Value of Data	Edited Result
X(10)/XX	ALPHANUMER01	ALPHANUMER/01
X(5)BX(7)	ALPHANUMERIC	ALPHA NUMERIC
A(5)BA(5)	ALPHABETIC	ALPHA BETIC
99,B999,B000	1234	01, 234, 000
99,999	12345	12,345

Figure 4-9. Examples of Simple Insertion Editing

Special Insertion Editing: This type of editing is valid only for numeric edited items.

The period is the special insertion symbol; it also represents the actual decimal point for alignment purposes.

The period insertion symbol is counted in the size of the item, and represents the position within the item where the actual decimal point will be inserted.

The actual decimal point and the assumed decimal point (the symbol V) must not both be specified in one PICTURE character-string.

Fixed Insertion Editing: This type of editing is valid only for numeric edited items. The following insertion symbols are used:

'CS' (currency symbol)

+ - CR DB (editing sign control symbols)

- In fixed insertion editing, only one currency symbol and one editing sign control symbol can be specified in one PICTURE character-string.
- Unless it is preceded by a + or - symbol, the currency symbol must be the leftmost character position in the character-string.
- When either + or - is used as a symbol, it must represent either the leftmost or rightmost character position in the character-string.
- When CR or DB is used as a symbol, it must represent the rightmost two character positions in the character-string.
- Editing sign control symbols produce results depending on the value of the data item as shown in Figure 4-10.

Examples of fixed insertion editing are shown in Figure 4-11.

Editing Symbol in PICTURE Character String	Resulting Data Item Positive or Zero	Resulting Data Item Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

Figure 4-10. Editing Sign Control Results

PICTURE	Value of Data	Edited Result
999.99+	+6555.556	555.55+
+9999.99	-6555.555	-6555.55
9999.99-	+1234.56	1234.56
$999.99	-123.45	$123.45
-$999.99	-123.456	-$123.45
$9999.99CR	+123.45	$0123.45
$9999.99DB	-123.45	$0123.45DB

Figure 4-11. Examples of Fixed Insertion Editing

Floating Insertion Editing: This type of editing is valid only for numeric edited items. The following symbols are used:

'CS' + -

Within one PICTURE character-string, these symbols are mutually exclusive as floating insertion characters.

Floating insertion editing is specified by using a string of at least two of the allowable floating insertion symbols to represent leftmost character positions in which these characters can be inserted.

The leftmost floating insertion symbol in the character-string represents the leftmost limit at which this character can appear in the data item. The rightmost floating insertion symbol represents the rightmost limit at which this character can appear.

The second leftmost floating insertion symbol in the character-string represents the leftmost limit at which numeric data can appear within the data item. Nonzero numeric data may replace all characters at or to the right of this limit.

Any simple insertion symbols (B 0 / ,) within or to the immediate right of the string of floating insertion symbols are considered part of the floating character-string. If the period special insertion symbol is included within the floating string, it is considered to be part of the character-string.

In a PICTURE character-string there are two methods to represent floating insertion editing and to perform editing:

1. Any or all leading numeric character positions to the left of the decimal point are represented by the floating insertion symbol. When editing is performed, a single floating insertion character is placed to the immediate left of the first nonzero digit in the data or of the decimal point, whichever is farther left. The character positions to the left of the inserted character are filled with spaces.

2. All the numeric character positions are represented by the floating insertion symbol. When editing is performed, then:
   a. If the value of the data is zero, the entire data item will contain spaces.
   b. If the value of the data is nonzero, the result is the same as in method 1.

To avoid truncation, the minimum size of the PICTURE character-string must be the sum of:

- The number of character positions in the sending item
- The number of nonfloating insertion symbols in the receiving item
- One character for the floating insertion symbol

Examples of floating insertion editing are shown in Figure 4-12.

PICTURE	Value of Data	Edited Result
$$$$.99	.123	$.12
$$$9.99	.12	$0.12
$$,$$$,999.99	-1234.56	$1,234.56
++,+++,999.99	-123456.789	-123,456.78
$$,$$$,$$$.99CR	-1234567.	$1,234,567.00CR
++,+++,+++,+++.	0000.00	

Figure 4-12. Examples of Floating Insertion Editing

Note: A single insertion symbol to the left of a simple or fixed insertion symbol followed by a string of floating insertion symbols is not considered part of the floating character-string. In the following example, the leftmost + in the character-string is considered to be a fixed insertion symbol and not a floating insertion symbol:

```
+,+++,+++.++
```

Zero Suppression and Replacement Editing: This type of editing is valid only for numeric edited items.

The symbols Z and * are used for zero suppression. These symbols are mutually exclusive in the PICTURE clause.

The following symbols are mutually exclusive as floating replacement symbols in one PICTURE character-string:

```
Z * + - 'CS' .
```

Zero suppression editing is specified by using a string of one or more of the allowable symbols to represent leftmost character positions in which zero suppression and replacement editing can be performed.

Any simple insertion symbols (B 0 / ,) within or to the immediate right of the string of floating editing symbols are considered part of the string. If the period special insertion symbol is included within the floating editing string, it is considered to be part of the character-string.

In a PICTURE character-string, there are two ways to represent zero suppression and perform editing:

- Any or all of the leading numeric character positions to the left of the decimal point are represented by suppression symbols. When editing is performed, any leading zero in the data that appears in the same character position as a suppression symbol is replaced by the replacement character. Suppression stops at the character farthest left that:
  - Does not correspond to a suppression symbol.
  - Contains nonzero data.
  - Is the decimal point.

- All the numeric character positions in the PICTURE character-string are represented by the suppression symbols. When editing is performed and the value of the data is nonzero, the result is the same as in the preceding rule. The following rules apply if the value of the data is zero:
  - If Z has been specified, the entire data item contains spaces.
  - If * has been specified, the entire data item, except the actual decimal point, contains asterisks.

  IBM Extension: The asterisk as a suppression symbol and the BLANK WHEN ZERO clause may be specified for the same entry. The asterisk overrides the BLANK WHEN ZERO clause if both are specified.

Examples of zero suppression and replacement editing are shown in Figure 4-13.

PICTURE	Value of Data	Edited Result
****.**	0000.00	****.**
ZZZZ.ZZ	0000.00	
ZZZZ.99	0000.00	.00
****.99	0000.00	****.00
ZZ99.99	0000.00	00.00
Z,ZZZ.ZZ+	+123.456	123.45+
*,***.**+	-123.45	**123.45-
,*,***.**+	+12345678.9	12,345,678.90+
$Z,ZZZ,ZZZ.ZZCR	+12345.67	$ 12,345.67
$B*,***,***.**BBDB	-12345.67	$ ***12,345.67 DB

Figure 4-13. Examples of Zero Suppression and Replacement Editing

Chapter 5. Procedure Division

PROCEDURE DIVISION CONCEPTS

The Procedure Division is required in every COBOL source program. The Procedure Division consists of optional Declaratives and procedures that contain the sections and/or paragraphs, sentences, and statements that solve a data processing problem.

Execution begins with the first statements in the Procedure Division, excluding Declaratives. Unless the logic flow indicates some other order, statements are executed in the order in which they are presented for compilation. The end of the Procedure Division and the physical end of the program is that physical position in a source program after which no further Procedure Division statements appear.

Declaratives

The Declarative section provides a method of invoking procedures that are executed when an exceptional condition occurs that is to be tested by the COBOL programmer.

When Declarative sections are specified, they must be grouped at the beginning of the Procedure Division. Declarative sections are preceded by the key word DECLARATIVES and followed by the key words END DECLARATIVES.

If Declarative sections are specified, the entire Procedure Division must be divided into sections.

Procedures

A *procedure* is a paragraph, group of paragraphs, a section, or a group of sections within the Procedure Division. A *procedure-name* is a user-defined name that identifies a section or a paragraph.

A *section* consists of a section header followed by zero, one, or more than one successive paragraphs. A *section-header* is a section-name followed by the key word SECTION, an optional priority-number, followed by a period and a space. Priority-numbers are explained under *Segmentation Feature* in Chapter 6. A *section-name* is a user-defined word that identifies a section. A section-name, because it cannot be qualified, must be unique. A section ends immediately before the next section header, at the end of the Procedure Division, or, in the Declaratives portion, at the key words END DECLARATIVES.

A *paragraph* consists of a paragraph-name followed by a period and a space. Zero, one, or more than one successive sentences are allowed. A *paragraph-name* is a user-defined word that identifies a paragraph. A paragraph-name, because it can be qualified, need not be unique. A paragraph ends immediately before the next paragraph-name or section header, at the end of the Procedure Division, or, in the Declaratives portion, at the key words END DECLARATIVES. If one paragraph in a program is contained within a section, then all paragraphs must be contained in sections.

A *sentence* consists of one or more statements terminated by a period and a space.

A *statement* is a syntactically valid combination of words (identifiers, figurative constants, and so on) and symbols (literals, relational-operators, and so on) beginning with a COBOL verb.

An *identifier* consists of the word or words necessary to make unique reference to a data item through qualification, subscripting, or indexing. In any Procedure Division reference except the class test, if the contents of an identifier are not compatible with the class specified through its PICTURE clause, results are unpredictable.

Note: A level-88 (condition-name) entry, because it is not a data item, cannot be an identifier. The associated conditional variable, however, can be an identifier.

PROCEDURE DIVISION ORGANIZATION

The structure of the Procedure Division is shown in the following formats.

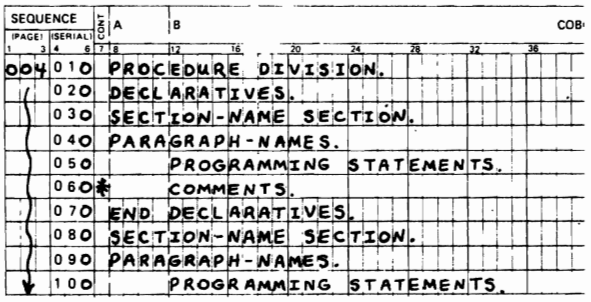
Format 1

```
PROCEDURE DIVISION [USING data-name-1 [,data-name-2] ...].
[ DECLARATIVES.
{section-name SECTION [segment-number] . declarative-sentence
[paragraph-name. [sentence] ...] ...} ...
  END DECLARATIVES.]
{section-name SECTION [segment-number] .
[paragraph-name. [sentence] ...] ...} ...
```

Format 2

```
PROCEDURE DIVISION [USING data-name-1 [,data-name-2] ...].
{paragraph-name. [sentence] ...} ...
```

Coding Example



SEQUENCE (PAGE)	SEQUENCE (SERIAL)	CONT	A	B
004	010			PROCEDURE DIVISION.
	020			DECLARATIVES.
	030			SECTION-NAME SECTION.
	040			PARAGRAPH-NAMES.
	050			PROGRAMMING STATEMENTS.
	060	*		COMMENTS.
	070			END DECLARATIVES.
	080			SECTION-NAME SECTION.
	090			PARAGRAPH-NAMES.
	100			PROGRAMMING STATEMENTS.

Categories of Sentences

There are three categories of sentences: conditional sentences, imperative sentences, and compiler-directing sentences.

A *conditional sentence* is a conditional statement, optionally preceded by an imperative statement, terminated by a period and a space.

An *imperative sentence* is an imperative statement, which may consist of a series of imperative statements, followed by a period and a space.

A *compiler-directing* sentence is a single compiler-directing statement, followed by a period and a space.

Categories of Statements

Three categories of statements are used in COBOL: conditional statements, imperative statements, and compiler-directing statements.

A *conditional statement* specifies that the truth value of a condition is to be determined, and that the subsequent action of the object program is dependent on this truth value. Figure 5-1 lists COBOL conditional statements.

An *imperative statement* specifies that an unconditional action is to be taken by the object program. An imperative statement may also consist of a series of imperative statements. Figure 5-2 lists COBOL imperative statements.

A *compiler-directing statement* causes the compiler to take a specific action during compilation. Figure 5-3 lists the COBOL compiler-directing statements.

Sample Procedure Division Statements

```
PROCEDURE DIVISION.
DECLARATIVES.
ERROR-IT SECTION.
    USE AFTER STANDARD   ERROR PROCEDURE ON INPUT-DATA.
ERROR-ROUTINE.
    IF CHECK-IT = '30' ADD 1 TO DECLARATIVE-ERRORS.
END DECLARATIVES.
BEGIN-NON-DECLARATIVES SECTION.
100-BEGIN-IT.
    OPEN INPUT INPUT-DATA OUTPUT REPORT-OUT.
110-READ-IT.
    READ  INPUT-DATA RECORD AT END MOVE 'Y' TO EOF-SW.
    ADD 1 TO RECORDS-IN.
200-MAIN-ROUTINE.
    PERFORM PROCESS-DATA UNTIL EOF-SW = 'Y'.
    PERFORM FINAL-REPORT THRU FINAL-REPORT-EXIT.
    DISPLAY 'TOTAL RECORDS IN = ' RECORDS-IN.
    DISPLAY 'DECLARATIVE ERRORS = >>> ' DECLARATIVE-ERRORS.
    STOP RUN.
PROCESS-DATA.
    IF RECORD-ID = 'G'
        PERFORM PROCESS-GEN-INFO
    ELSE
        IF RECORD-CODE = 'C'
            PERFORM PROCESS-SALES-DATA
        ELSE
            PERFORM UNKNOWN-RECORD-TYPE.
```

Decision	IF
Input/Output	DELETE....INVALID KEY READ....AT END READ...INVALID KEY REWRITE...INVALID KEY START...INVALID KEY WRITE...AT END-OF-PAGE WRITE...INVALID KEY
Arithmetic	ADD...ON SIZE ERROR COMPUTE...ON SIZE ERROR DIVIDE...ON SIZE ERROR MULTIPLY...ON SIZE ERROR SUBTRACT...ON SIZE ERROR
Procedure Branching	PERFORM...UNTIL
Data Movement	STRING...ON OVERFLOW UNSTRING...ON OVERFLOW
Table Handling	SEARCH
Ordering	RETURN...AT END
Debug	EXHIBIT...CHANGED

Figure 5-1. Conditional Statements and Their Categories

Arithmetic	ADD[1]
	COMPUTE[1]
	DIVIDE[1]
	INSPECT(TALLYING)
	MULTIPLY[1]
	SUBTRACT[1]
Data Movement	ACCEPT (DATE, DAY, TIME)
	INSPECT (REPLACING)
	MOVE
	STRING[3]
	UNSTRING[3]
Ending	EXIT PROGRAM
	STOP RUN
Input/Output	ACCEPT (mnemonic)
	ACQUIRE
	CLOSE
	DELETE[2]
	DISPLAY
	DROP
	OPEN
	READ[4]
	REWRITE[2]
	SET[6]
	START[2]
	STOP literal
	WRITE[5]

[1]Without the SIZE ERROR option
[2]Without the INVALID KEY option
[3]Without the ON OVERFLOW option
[4]Without the AT END or INVALID KEY options
[5]Without the INVALID KEY or END-OF-PAGE options
[6]When used to modify external switch values

Figure 5-2 (Part 1 of 2). Categories of Imperative Statements

Ordering	MERGE
	RELEASE
	RETURN
	SORT
Procedure Branching	ALTER
	CALL
	EXIT
	GO
	PERFORM
Table Handling	SET
Subprogram Linkage	CALL
Debug	EXHIBIT
	READY TRACE
	RESET TRACE

Figure 5-2 (Part 2 of 2). Categories of Imperative Statements

Library	COPY
Declarative	USE
Documentation	ENTER

Figure 5-3. Categories of Compiler-Directing Statements

Categories of Expressions

Two categories of expressions are used in COBOL. arithmetic expressions and conditional expressions.

Arithmetic expressions are used as operands of conditional or arithmetic statements. Any arithmetic expression may be preceded by an unary operator.

A conditional expression causes the object program to select alternative paths of control, depending on the value of a truth test. There are two types of conditional expressions: simple conditions and complex conditions. Conditional expressions can be specified in the IF, PERFORM, and SEARCH statements.

ARITHMETIC EXPRESSIONS

Arithmetic expressions are used as operands of certain conditional and arithmetic statements. An arithmetic expression may consist of any of the following:

1. An identifier described as a numeric elementary item.
2. A numeric literal.
3. Identifiers and literals, as defined in Items 1 and 2, separated by arithmetic operators.
4. Two arithmetic expressions, as defined in Items 1, 2, and/or 3, separated by an arithmetic operator.
5. An arithmetic expression, as defined in Items 1, 2, 3, and/or 4, enclosed in parentheses.

Any arithmetic expression may be preceded by a unary operator.

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

Arithmetic Operators

The five binary arithmetic operators and two unary arithmetic operators shown in Figure 5-4 may be used in arithmetic expressions. The arithmetic operators are represented by specific characters that must be preceded and followed by a space.

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

Unary Operator	Meaning
+	Multiplication by +1; retains original sign
-	Multiplication by -1; changes sign

Figure 5-4. Binary and Unary Operators

Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first. When expressions are contained within a nest of parentheses, evaluation proceeds from the least inclusive to the most inclusive set.

When parentheses are not used, or when parenthesized expressions are at the same level of inclusiveness, the following hierarchical order is implied:

1. Unary operator
2. Exponentiation
3. Multiplication and division
4. Addition and subtraction

When exponentiation is used as an arithmetic operator, the exponent identifier or literal must be a positive integral value.

Parentheses either eliminate ambiguities in logic where consecutive operations appear at the same hierarchical level or modify the normal hierarchical sequence of execution when this is necessary. When the order of consecutive operations at the same hierarchical level is not completely specified by parentheses, the order is from left to right.

Figure 5-5 shows permissible arithmetic symbol pairs. An arithmetic symbol pair is the appearance of two such symbols in sequence.

An arithmetic expression may begin only with a left parenthesis, a unary operator, or a variable (that is, an identifier or literal). An arithmetic expression may end only with a right parenthesis or a variable. An arithmetic expression must contain at least one reference to an identifier or literal. There must be a one-to-one correspondence between left and right parentheses in an arithmetic expression; each left parenthesis is placed to the left of its corresponding right parenthesis.

First Symbol \ Second Symbol	Variable (identifier or literal)	* / ** + -	unary + unary -	(	)
Variable (identifier or literal)	-	p	-	-	p
* / ** + -	p	-	p	p	-
unary + or unary -	p	-	-	p	-
(	p	-	p	p	-
)	-	p	-	-	p

p indicates a permissible pairing
- indicates that the pairing is not permitted

Figure 5-5. Valid Arithmetic Symbol Pairs

CONDITIONAL EXPRESSIONS

A conditional expression causes the object program to select alternative paths of control, depending on the truth value of a test. Conditional expressions can be specified in IF, PERFORM, and SEARCH statements. A conditional expression can be specified in simple conditions and in complex conditions. Both simple and complex conditions can be enclosed within any number of paired parentheses; parentheses do not change the category of the condition.

Simple Conditions

There are five simple conditions: class condition, condition-name condition, relation condition, sign condition, and switch-status condition. A simple condition has a truth value of true or false. When a simple condition is enclosed in paired parentheses, its truth value is not changed.

Class Condition

The class condition determines whether a data item is alphabetic or numeric.

Format

identifier IS [NOT] { NUMERIC / ALPHABETIC }

The identifier being tested must be described implicitly or explicitly as USAGE DISPLAY. The identifier is determined to be numeric only if the contents consist of any combination of the digits 0 through 9, with or without an operational sign.

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

If the PICTURE of the identifier being tested does contain an operational sign, the identifier is determined to be numeric only if the item is an elementary item, the contents are numeric, and a valid operational sign is present.

In the EBCDIC collating sequence, valid embedded operational signs are hex F and D. For items described with the SIGN IS SEPARATE clause, valid operational signs are + (hex 4E) and - (hex 60).

The NUMERIC test cannot be used with an identifier described either as alphabetic or as a group item that contains one or more signed elementary items. The identifier being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters A through Z and the space.

IBM Extension: The numeric class test can also be specified for an identifier that is defined as USAGE IS COMPUTATIONAL.

The ALPHABETIC test cannot be used with an identifier described as numeric.

Figure 5-6 shows valid forms of the class test.

Type of Identifier	Valid Forms of the Class Test
Alphabetic	ALPHABETIC NOT ALPHABETIC
Alphanumeric	ALPHABETIC NOT ALPHABETIC NUMERIC NOT NUMERIC
Zoned Decimal	NUMERIC NOT NUMERIC

Figure 5-6. Valid Forms of the Class Test

Condition-Name Condition

A condition-name condition causes a conditional variable to be tested to determine whether its value is equal to any of the values associated with the condition-name (level-88 item).

Format

condition-name

A condition-name is used in conditions as an abbreviation for the relation condition, because the specified condition-name is equal to only one of the values or ranges of values assigned to the specified conditional variable. The result of the test is true if one of the values corresponding to the condition-name equals the current value of the associated conditional variable.

If the condition-name is associated with a range of values or with several ranges of values, the conditional variable is tested to determine whether or not its value falls within the range(s), including the end values. The result of the test is true if one of the values corresponding to the condition-name equals the value of its associated conditional variable.

The following example illustrates the usage of condition-names and conditional variables:

```
02  GRADE-ID PIC 99.
    88 PRIMARY-OTHER    VALUE 1 THRU 3, 5, 6.
    88 PRIMARY-FOUR     VALUE 4.
    88 JUNIOR-HI        VALUE 7 THROUGH 9.
    88 SENIOR-HI        VALUE 10 THROUGH 12.
```

GRADE-ID is the conditional variable, PRIMARY-OTHER, PRIMARY-FOUR, JUNIOR-HI, and SENIOR-HI are condition-names. For individual records in the file, only one of the values specified in the condition-name entries can be present. To determine the grade level of a specific record, any of the following can be coded:

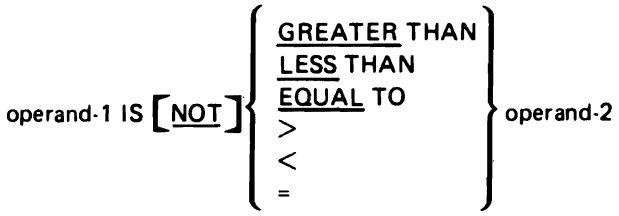
```
IF PRIMARY-OTHER...
  (which tests for values 1, 2, 3, 5, 6)
IF PRIMARY-FOUR...
  (which tests for value 4)
IF JUNIOR-HI...
  (which tests for values 7, 8, 9)
IF SENIOR-HI...
  (which tests for values 10, 11, 12)
```

Depending on the evaluation of the condition-name condition, alternative paths of execution are taken by the object program.

Relation Condition

A relation condition causes a comparison between two operands, either of which may be an identifier, a literal, or an arithmetic expression.

Format



Operand-1 is the subject of the relation condition; operand-2 is the object of the relation condition. Operand-1 and operand-2 may each be an identifier, a literal, or an arithmetic expression. The relation condition must contain at least one reference to an identifier. Except when two numeric operands are compared, operand-1 and operand-2 must have the same USAGE.

The relational operator specifies the type of comparison to be made. Figure 5-7 shows relational operators and their meanings. Each relational operator must be preceded and followed by a space.

Relational Operator	Meaning
IS [NOT] GREATER THAN IS [NOT] >	Greater than or not greater than
IS [NOT] LESS THAN IS [NOT] <	Less than or not less than
IS [NOT] EQUAL TO IS [NOT] =	Equal to or not equal to

Figure 5-7. Relational Operators and Their Meanings

IBM Extension:

Boolean Considerations: The valid types of relation conditions that can be used with Boolean data items are EQUAL TO and NOT EQUAL TO.

Rules for numeric and nonnumeric comparisons are given in the following paragraphs. If either of the operands is a group item, nonnumeric comparison rules apply. Figure 5-8 summarizes the permissible comparisons.

First Operand	Second Operand													
	BO	GR	AL	AN	ANE	NE	FC[1] NNL	ZR NL	ZD	BI	PD	AE	IN	IDI
Group (GR)		NN	NN	NN	NN	NN	NN	NN	NN					
Alphabetic (AL)		NN	NN	NN	NN	NN	NN	NN	NN					
Alphanumeric (AN)		NN	NN	NN	NN	NN	NN	NN	NN					
Alphanumeric edited (ANE)		NN	NN	NN	NN	NN	NN	NN	NN					
Numeric edited (NE)		NN	NN	NN	NN	NN	NN	NN	NN					
Figurative constant (FC)[1] and nonnumeric literal (NNL)		NN	NN	NN	NN	NN			NN					
Figurative constant ZERO (ZR) and numeric literal (NL)		NN	NN	NN	NN	NN			NU	NU	NU	NU	IO[2]	
Zoned decimal (ZD)		NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU	IO[2]	
Binary (BI)								NU	NU	NU	NU	NU	IO[2]	
Packed decimal (PD)								NU	NU	NU	NU	NU	IO[2]	
Arithmetic expression (AE)								NU	NU	NU	NU	NU		
Index-name (IN)								IO[2]	IO[2]	IO[2]	IO[2]		IO	IV
Index data item (IDI)													IV	IV
Boolean	BO													

NN = comparison as described for nonnumeric operands.
NU = comparison as described for numeric operands.
IO = comparison as described for two index-names or index data items.
IV = comparison as described for index data items.
BO = Boolean data items.

[1] FC includes all figurative constants except ZERO.
[2] Valid only if the numeric item is an integer.

Figure 5-8. Permissible Comparisons of Operands

Comparison of Numeric Operands: For numeric class operands, algebraic values are compared. The length (number of digits) of the operands is not significant. Zero is considered a unique value, regardless of the sign. Unsigned numeric operands are considered positive; regardless of their USAGE, comparison of numeric operands is permitted.

Comparison of Nonnumeric Operands: A comparison of two nonnumeric operands or of one numeric and one nonnumeric operand is made with respect to the binary collating sequence of the character set in use.

When a nonnumeric and a numeric operand are compared, the following rules apply:

- If the nonnumeric operand is a literal or an elementary data item, the numeric operand is treated as though it were moved to an alphanumeric elementary data item of the same size, and the contents of this alphanumeric data item were then compared with the nonnumeric operand.

- If the nonnumeric operand is a group item, the numeric operand is treated as though it were moved to a group item of the same size, and the contents of this group item were then compared with the nonnumeric operand. For further discussion of the rules for alphanumeric and group move operations, see the *MOVE Statement* in this chapter.

Numeric and nonnumeric operands may be compared only when their USAGE, explicitly or implicitly, is the same. In such comparisons, the numeric operand must be described as an integer literal or data item; noninteger literals and data items must not be compared with nonnumeric operands.

The size of each operand is the total number of characters in that operand; the size affects the result of the comparison. There are two kinds of operands to consider: operands of equal size and operands of unequal size.

Operands of Equal Size: Characters in corresponding positions of the two operands are compared, beginning with the leftmost character and continuing through the rightmost character.

If all pairs of characters through the last pair test as equal, the operands are considered equal. If a pair of unequal characters is encountered, the characters are tested to determine their relative positions in the collating sequence. The operand containing the character higher in the sequence is considered the greater operand.

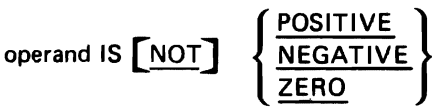
Operands of Unequal Size: If the operands are of unequal size, the comparison is made as though the shorter operand were extended to the right with enough spaces to make the operands equal in size.

Note: Valid comparisons for index-names and index data items are discussed under *Table Handling* in Chapter 6.

Sign Condition

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

Format



The operand being tested must be defined as a numeric identifier or as an arithmetic expression that contains at least one reference to an identifier.

The operand is POSITIVE if its value is greater than zero, NEGATIVE if its value is less than zero, and ZERO if its value is equal to zero. An unsigned operand is POSITIVE or ZERO.

When NOT is specified, one algebraic test is executed for the truth value of the sign condition. For example, NOT ZERO is regarded as true when the operand tested is positive or negative in value.

Switch-Status Condition

The switch-status condition determines the on or off status of an UPSI switch.

Format

condition-name

The condition-name must be defined to be associated with the ON or OFF value of a switch in the SPECIAL-NAMES paragraph.

The switch-status condition tests the value associated with the condition-name. The result of the test is true if the UPSI switch is set to the position corresponding to condition-name.

Complex Conditions

A complex condition is a condition in which one or more logical operators act upon one or more conditions. Complex conditions include:

- Negated simple conditions
- Combined conditions
- Negated combined conditions

Each logical operator must be preceded and followed by a space. The logical operators and their meanings are shown in Figure 5-9.

Logical Operator	Meaning
AND	Logical conjunction–the truth value is true when both conditions are true.
OR	Logical inclusive OR–the truth value is true when either or both conditions are true.
NOT	Logical negation–reversal of truth value (the truth value is true if the condition is false).

Figure 5-9. Logical Operators and Their Meanings

Negated Simple Conditions

A simple condition is negated through the use of the logical operator NOT.

Format

NOT simple-condition

The simple-condition to be negated can be a class condition, a condition-name condition, a relation condition, a sign condition, or a switch-status condition. The simple-condition may not be negated if the condition itself contains a NOT.

The negated simple-condition gives the opposite truth value as the simple condition. That is, if the truth value of the simple-condition is true, then the truth value of that same negated simple-condition is false.

Placing a negated simple-condition within parentheses does not change its truth value. For example, the following two statements are equivalent:

NOT A IS EQUAL TO B.

NOT (A IS EQUAL TO B).

Combined Conditions

Two or more conditions can be logically connected to form a combined condition.

Format

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

The condition to be combined can be a simple-condition, a negated simple-condition, a combined condition, a negated combined condition (that is, the NOT logical operator followed by a combined condition enclosed in parentheses). Combinations of the preceding conditions are specified according to the rules given in Figure 5-10.

Parentheses are never needed when either AND or OR (but not both) are used exclusively in one combined condition. However, parentheses may be needed to find a final truth value when a combination of AND, OR, and NOT is used. There must be a one-to-one correspondence between left and right parentheses with each left parenthesis to the left of its corresponding right parenthesis.

Figure 5-10 summarizes the way in which conditions and logical operators can be combined and parenthesized. Figure 5-11 illustrates the relationships between logical operators and conditions C1 and C2 where C1 and C2 are any conditions as defined above.

Condition Element	Permissible Position in Conditional Expressions			
	Leftmost	When Not Leftmost May Be Immediately Preceded By:	When Not Rightmost May Be Immediately Followed By:	Rightmost
simple-condition	yes	OR NOT AND (	OR AND)	yes
OR AND	no	simple-condition)	simple-condition NOT (	no
NOT	yes	OR AND (	simple-condition (	no
(	yes	OR NOT AND (	simple-condition NOT (	no
)	no	simple-condition)	OR AND)	yes

Figure 5-10. Valid Combinations of Conditions, Logical Operators, and Parentheses in a Conditional Expression

Values for C1	Values for C2	C1 AND C2	C1 OR C2	NOT (C1 AND C2)	NOT C1 AND C2	NOT (C1 OR C2)	NOT C1 OR C2
True	True	True	True	False	False	False	True
False	True	False	True	True	True	False	True
True	False	False	True	True	False	False	False
False	False	False	False	True	False	True	True

Figure 5-11. How Logical Operators Affect the Evaluation of Conditions

The truth value of a complex condition depends on the truth values of the simple conditions and negated simple conditions that make up the complex condition. The logical operators tell the compiler how to combine these individual truth values.

Evaluating Conditional Expressions: If parentheses are used, logical evaluation of combined conditions proceeds in the following order:

1. Conditions within parentheses are evaluated first.

2. Within nested parentheses, evaluation proceeds from the least inclusive condition to the most inclusive condition.

If parentheses are not used (or are not at the same level of inclusiveness), the combined condition is evaluated in the following order:

1. Arithmetic expressions

2. Simple-conditions in the following order:
   a. Relation
   b. Class
   c. Condition-name
   d. Switch-status
   e. Sign

3. Negated simple-conditions in the same order as item 2.

4. Combined conditions, in the following order:
   a. AND
   b. OR

5. Negated combined conditions in the following order:
   a. AND
   b. OR

6. Consecutive operands at the same evaluation-order level are evaluated from left to right

For example:

```
A IS NOT GREATER THAN B OR A + B IS EQUAL
TO C AND D IS POSITIVE
```

This expression is evaluated as if it were enclosed in parentheses as follows:

```
(A IS NOT GREATER THAN B) OR (((A+B) IS EQUAL
TO C) AND (D IS POSITIVE)).
```

The order of evaluation is as follows:

1. (A IS NOT GREATER THAN B) is evaluated, giving some intermediate truth value; for example, t1.

2. (A + B) is evaluated, giving some intermediate result; for example, x.

3. (x IS EQUAL TO C) is evaluated, giving some intermediate truth value; for example, t2.

4. (D IS POSITIVE) is evaluated, giving some intermediate truth value; for example, t3.

5. (t2 AND t3) is evaluated, giving some intermediate truth value; for example, t4.

6. (t1 OR t4) is evaluated, giving the final truth value, and the result of the expression.

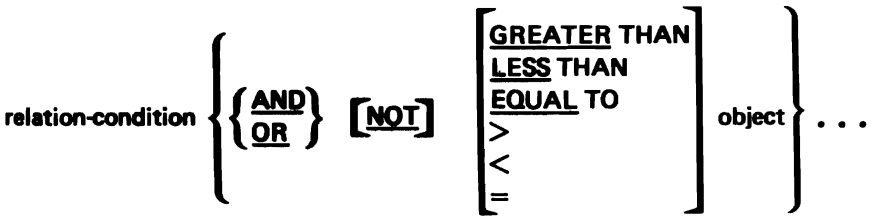
Programming Note: Every condition in the expression will always be evaluated before a final truth value is determined. The user must ensure that any subscripted or indexed data items stay within the described bounds of the table.

Abbreviated Combined Relation Conditions

When relation-conditions are written consecutively and no parentheses are used within the consecutive sequence, any relation-condition after the first can be abbreviated by either:

- Omission of the subject
- Omission of the subject and relational operator

Format



In any consecutive sequence of relation-conditions, both forms of abbreviation can be specified. The abbreviated condition is evaluated as if:

- The last stated subject is the missing subject.
- The last stated relational operator is the missing relational operator.
- The resulting combined condition must comply with the rules for element sequence in combined conditions, as shown in Figure 5-10.
- The word NOT is considered part of the relational operator in the forms NOT GREATER THAN, NOT >, NOT LESS THAN, NOT <, NOT EQUAL TO, and NOT =.
- NOT in any other position is considered a logical operator, and thus results in a negated relation-condition.

Figure 5-12 shows examples of abbreviated combined relation-conditions and their nonabbreviated equivalents.

Abbreviated Combined Relation-Condition	Nonabbreviated Equivalent
A = B AND NOT LESS THAN C OR D	((A = B) AND (A NOT LESS THAN C)) OR (A NOT LESS THAN D)
A NOT GREATER THAN B OR C	(A NOT GREATER THAN B) OR (A NOT GREATER THAN C)
NOT A = B OR C	(NOT (A = B) OR (A = C))
NOT (A = B OR LESS THAN C)	NOT ((A = B) OR (A LESS THAN C))
NOT (A NOT = B AND C AND NOT D)	NOT (((A NOT = B) AND (A NOT = C)) AND (NOT (A NOT = D)))

Figure 5-12. Abbreviated Combined Relation-Condition Equivalent

DECLARATIVES

The Declaratives section provides a method of invoking procedures that are executed when an exceptional condition occurs that cannot normally be tested by the COBOL programmer. Declarative procedures are provided for the processing of exceptional input/output conditions and debugging procedures.

Format

```
PROCEDURE DIVISION [USING data-name-1 [,data-name-2 ] . . . ] .

[ DECLARATIVES.

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

END DECLARATIVES. ]
```

Declarative procedures are written at the beginning of the Procedure Division in a series of Declarative sections. Each such section is preceded by a USE sentence that identifies the function of this section. The series of procedures that follow specify what actions are to be taken when the exceptional condition occurs. Each Declarative section ends with the occurrence of another section-name followed by a USE sentence, or with the key words END DECLARATIVES.

The entire group of Declarative procedures is preceded by the key word DECLARATIVES, written on the next line after the Procedure Division header; the group is followed by the key words END DECLARATIVES. The key words DECLARATIVES and END DECLARATIVES must each begin in Area A and be followed by a period. No other text may appear on the same line.

In the Declaratives portion of the Procedure Division, each section header (with an optional segment number) must be followed by a period and a space, and must be followed by a USE sentence followed by a period and a space. No other text may appear on the same line. There are two forms of the USE sentence:

- USE AFTER EXCEPTION/ERROR
- USE FOR DEBUGGING

The USE sentence itself is never executed; instead, the USE sentence defines the conditions that will cause execution of the immediately following procedural paragraphs, which specify the actions to be taken. After the procedure is executed, control is returned to the routine that activated it.

Within a Declarative procedure, except for the USE statement itself, there must be no reference to any nondeclarative procedure.

Within a Declarative procedure, no statement may be executed that would cause execution of a USE procedure that has been previously invoked and has not yet returned control to the invoking routine.

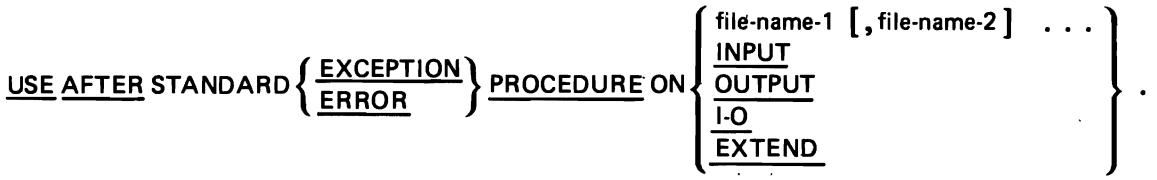
An exit from a Declarative procedure is effected by executing the last statement in the procedure.

In this chapter, only the USE AFTER EXCEPTION/ERROR procedure is described. The USE FOR DEBUGGING procedure is described under *Debugging Features* in Chapter 6.

EXCEPTION/ERROR Declarative

The EXCEPTION/ERROR Declarative specifies procedures for input/output exception or error handling that are to be executed in addition to the standard system procedures.

Format



The words EXCEPTION and ERROR are synonymous and may be used interchangeably.

File-Name Option

This option is valid for sequential, indexed, relative, and TRANSACTION files. When this option is specified, the procedure is executed only for the file(s) named. No file-name can refer to a sort-merge file. For any given file, only one EXCEPTION/ERROR procedure may be specified. For example, if an input file is specifically named in one EXCEPTION/ERROR procedure, there must not also be an EXCEPTION/ERROR procedure for all INPUT files.

INPUT Option

This option is valid for sequential, indexed, and relative files. When this option is specified, the procedure is applicable to all files opened in INPUT mode.

OUTPUT Option

This option is valid for sequential, indexed, and relative files. When this option is specified, the procedure is applicable to all files opened in OUTPUT mode.

I-O Option

This option is valid for sequential, indexed, relative, and TRANSACTION files. When this option is specified, the procedure is applicable to all files opened in I-O mode.

EXTEND Option

This option is valid for sequential files only. When this option is specified, the procedure is applicable to all files opened in EXTEND mode.

General Considerations

The EXCEPTION/ERROR procedure is executed when one of the following conditions exists:

- After completing the standard system input/output error routine.
- Upon recognition of an INVALID KEY or AT END condition when an INVALID KEY or AT END option has not been specified in the input/output statement.
- When Status Key 1 is not equal to 0 following an I/O operation.

After execution of the EXCEPTION/ERROR procedure, control is returned to the statement immediately following the input/output statement that caused the error.

The EXCEPTION/ERROR procedures are performed when an input/output error occurs during execution of a READ, WRITE, REWRITE, START, DELETE, OPEN, CLOSE, ACQUIRE, or DROP statement. For example, these procedures are activated when an input/output statement fails on a file that is in the open status.

The EXCEPTION/ERROR procedures are not performed when the following conditions exist:

- An OPEN statement fails on a file that is not in the open status.
- A CLOSE statement fails because the file is in the close status.
- Any input/output statement fails because the file has not been opened.

Within a Declarative procedure, there must be no reference to any nondeclarative procedure. In the nondeclarative portion of the program, there must be no reference to procedure-names that appear in an EXCEPTION/ERROR Declarative procedure, except that PERFORM statements may refer to an EXCEPTION/ERROR procedure or to procedures associated with it.

Within an EXCEPTION/ERROR Declarative procedure, no statement may be executed that causes execution of a USE procedure that has been previously invoked and has not yet returned control to the invoking routine.

IBM Extension:

TRANSACTION File Considerations

In an EXCEPTION/ERROR Declarative for the TRANSACTION file, only the file-name or I-O options are allowed. All other options and all rules are the same as those for any EXCEPTION/ERROR Declarative for any file.

Programming Notes

EXCEPTION/ERROR procedures can be used to check the status key values whenever an input/output error occurs.

Care should be used in specifying EXCEPTION/ERROR procedures for any file. Prior to successful completion of an initial OPEN for any file, the current Declarative has not yet been established by the object program. Therefore, if any other I/O statement is executed for a file that has never been opened,no Declarative can receive control. However, if this file has been previously opened, the last previously established Declarative procedure receives control.

For example, an OPEN OUTPUT statement establishes a Declarative procedure for this file, and the file is then closed without error. During later processing, if a logic error occurs, control will go to the Declarative procedure established when the file was opened OUTPUT.

CONDITIONAL STATEMENTS

A conditional statement specifies that a truth value of a condition is to be determined, and that the subsequent action of the object program depends on this truth value. Figure 5-1 gives a list of the conditional statements.

Only the IF statement is discussed in this section; the other conditional statements are discussed elsewhere in this manual.

IF Statement

The IF statement causes a condition to be evaluated, and provides for alternative actions in the object program, depending on that value.

Format

```
IF condition THEN { statement-1      } [ { ELSE statement-2      } ]
                  { NEXT SENTENCE    }   { ELSE NEXT SENTENCE   }
```

Statement-1 or statement-2 can be any one of the following:

- An imperative statement
- A conditional statement
- An imperative statement followed by a conditional statement

If the condition tested is true, one of the following actions takes place:

- Statement-1, if specified, is executed. If statement-1 contains a procedure branching statement, control is transferred according to the rules for that statement. If statement-1 does not contain a procedure-branching statement, the ELSE phrase, if specified, is ignored, and control passes to the next executable sentence.
- NEXT SENTENCE, if specified, is executed; that is, the ELSE phrase, if specified, is ignored, and control passes to the next executable sentence.

If the condition tested is false, one of the following actions take place:

- ELSE statement-2, if specified, is executed. If statement-2 contains a procedure-branching statement, control is transferred according to the rules for that statement. If statement-2 does not contain a procedure-branching statement, control is passed to the next executable sentence.
- ELSE NEXT SENTENCE, if specified, is executed. Therefore, statement-1, if specified, is ignored; control passes to the next executable sentence.
- If ELSE clause is omitted, control passes to the next executable sentence.
- The ELSE NEXT SENTENCE phrase can be omitted if it immediately precedes the period that ends the conditional sentence.

Note: When the ELSE clause is omitted, all statements following the condition and preceding the period for the sentence are considered to be part of statement-1.

IBM Extension: `THEN is accepted and ignored if present.`

Nested IF Statements

The presence of one or more IF statements within their initial IF statement constitutes a nested IF statement.

Statement-1 and statement-2 in IF statements can consist of one or more imperative statements and/or a conditional statement. If an IF statement appears as statement-1 or as part of statement-1, it is said to be nested. Nesting statements is much like specifying subordinate arithmetic expressions enclosed in parentheses and combined in larger arithmetic expressions.

IF statements contained within IF statements must be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered must be considered to apply to the immediately preceding IF that has not already paired with an ELSE.

Figure 5-13 shows the possible true/false combinations for the following nested IF statement:

```
IF condition-1
   statement-1-1
   IF condition-2
      IF condition-3
         statement-3-1
      ELSE
         statement-3-2
   ELSE
      statement-2-2
      IF condition-4
         IF condition-5
            statement-5-1
         ELSE
            statement-5-2.
```

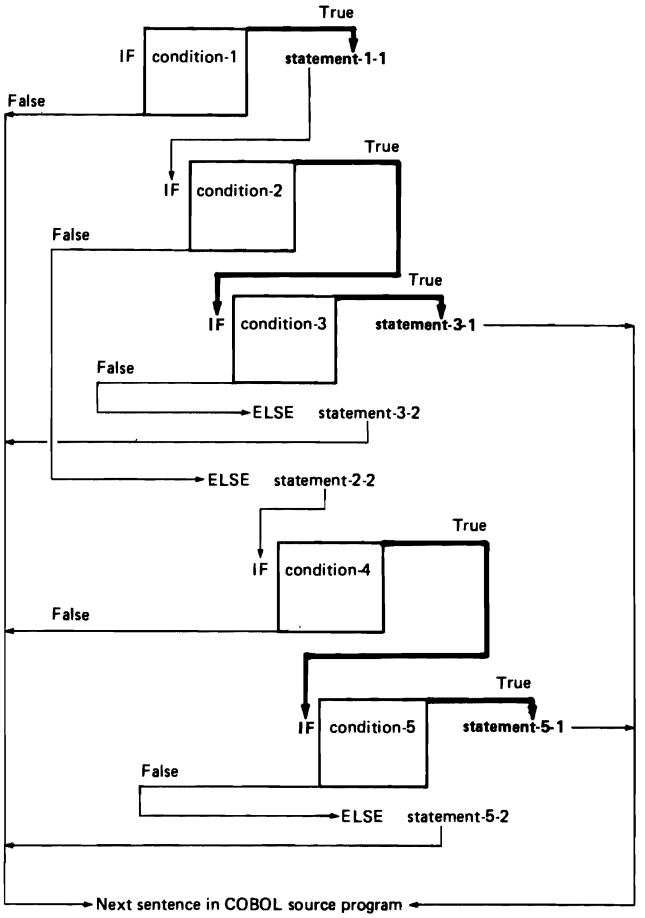


Figure 5-13. Nested IF Statement–True/False Combinations

Programming Notes: Because their logic is often difficult to follow, nested IF statements should, wherever possible, be avoided in a COBOL program. Often a series of simple IF statements can be used in place of the nested IF statement.

For example, the following series of simple IF statements give results equivalent to those achieved using the preceding nested IF statement example:

```
    IF condition-1 NEXT SENTENCE
       ELSE GO TO PARA-2.

    statement-1-1.

    IF condition-2 NEXT SENTENCE
       ELSE GO TO PARA-1.

    IF condition-3 statement-3-1 GO TO PARA-2
       ELSE statement-3-2 GO TO PARA-2.

PARA-1.

    statement-2-2.

    IF condition-4 NEXT SENTENCE
       ELSE GO TO PARA-2.

    IF condition-5 statement-5-1
       ELSE statement-5-2.

PARA-2.

    next-executable-statement.
```

Notice that Figure 5-13 also illustrates the logic flow for the preceding series of simple IF statements.

INPUT/OUTPUT STATEMENTS

COBOL input/output statements transfer data to and from files. In COBOL, the unit of data made available to the program is a record, and the COBOL programmer need concern himself only with such records. Provision is automatically made for such operations as the movement of data into buffers and/or internal storage, validity checking, error correction (when feasible), and unblocking and blocking of records.

The description of the file in the Environment Division and the Data Division governs which input/output statements are allowed in the Procedure Division.

There is special processing for deleted records (deleted records are valid only for relative and index files), and there are certain restrictions when using deleted records. For a full explanation of the limitations associated with deleted record processing see *Indexed and Relative File Contents* in Chapter 8.

Common Options

There are several options common to input/output statements. These are: status key, INVALID KEY condition, INTO/FROM identifier option, and current record pointer. The description of these options precedes the descriptions of the individual statements.

Status Key—General Considerations

If the FILE STATUS clause is specified in the file-control entry, a value is placed in the specified status key (the 2-character data item named in the FILE STATUS clause) during execution of any request on that file; the value indicates the status of that request. The value is placed in the status key before execution of any EXCEPTION/ERROR Declarative or INVALID KEY/AT END option associated with the request.

The first character of the status key is known as status key 1; the second character is known as status key 2. Combinations of possible values and their meanings are shown in Figure 5-14 and Appendix H.

IBM Extension:

TRANSACTION File Extended File Status Key: The extended file status key for a TRANSACTION file is four characters long. Characters 1 and 2 contain the ICF major return code; characters 3 and 4 contain the ICF minor return code. ICF return codes are described in the ICF Reference Manual.

Status Key 1	Status Key 2	Meaning
0		Successful completion
	0	No further information
	1	Initial READ from a REQUESTOR (IBM Extension)
1	0	At end of file (no outstanding invites)
2		Invalid key
	1	Sequence error
	2	Duplicate key when duplicates are not allowed
	3	No record found
	4	Boundary violation–indexed or relative file
3		Permanent error
	0	No further information
	4	Boundary violation–sequential file
9		Other errors (IBM Extensions)
	0	Invalid update, add, or output operation
	1	Undefined access type
	2	Logic error (I/O to unopened file, file locked, already OPEN, already CLOSED, or invalid operation)
	4	No current record pointer for I/O request
	5	Invalid or incomplete file information
	7	Invalid Op Code
	9	Undefined
	A	STOP requested by system operator
	C	Acquire operation failed, terminal not in standby mode
	D	Terminal operator released work station with INQUIRY key
	E	SRT program released its requestor, I/O rejected
	F	Acquire operation failed, either operator signed on is unauthorized or program is unauthorized to use resources
	G	Input data rejected, buffer too small
	H	Acquire operation failed, resource is unavailable or currently owned by another program
	I	Write operation failed, input data already received by Data Management
	N	Temporary error (error during session)

Figure 5-14. Status Key Values and Meanings

INVALID KEY Condition

The INVALID KEY condition can occur during execution of a START, READ, WRITE, REWRITE, or DELETE statement. When the INVALID KEY condition is recognized, the actions are taken in the following order:

1. If the FILE-STATUS clause is specified in the file-control entry, a value is placed into the status key to indicate an INVALID KEY condition (see Figure 5-14).

2. If the INVALID KEY option is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative-statement. Any EXCEPTION/ERROR declarative procedure specified for this file is not performed.

3. If the INVALID KEY option is not specified, but an EXCEPTION/ERROR declarative procedure is specified for the file, the EXCEPTION/ERROR procedure is executed.

When an INVALID KEY condition occurs, the input/output statement that caused the condition is unsuccessful. If the INVALID KEY option is not specified for a file, an EXCEPTION/ERROR procedure must be specified.

INTO/FROM Identifier Option

This option is valid for READ, REWRITE, and WRITE statements. The identifier specified must be the name of an entry in the Working-Storage Section, the Linkage Section, or of a record description for another previously opened file. Record-name/file name and identifier must not refer to the same storage area. In both options, an implicit move is executed according to MOVE statement rules without the CORRESPONDING option.

The following illustrates the use of the INTO/FROM identifier option in an input/output statement:

```
READ file-name RECORD INTO identifier.

WRITE record-name FROM identifier.
```

Current Record Pointer

The current record pointer identifies which record will be accessed by a sequential input request. The record identified depends on the statement being executed. The OPEN, READ, and START statements position the current record pointer as follows:

- The OPEN statement positions the current record pointer at the first record in the file.

- For a sequential READ statement, the following considerations apply:
  - If an OPEN or a START statement positioned the current record pointer, the record identified by the current record pointer is made available.
  - If a previous READ statement positioned the current record pointer, the current record pointer is updated to point to the next existing record in the file; that record is then made available.

- The START statement positions the current record pointer at the first record in the file that satisfies the implicit or explicit comparison specified in the START statement.

The setting of the current record pointer is affected only by the OPEN, START, RETURN, and READ statements. The concept of the current record pointer has no meaning for random access files, TRANSACTION files, or output files.

The current record pointer is not used for random retrieval of input records or for output files.

ACCEPT Statement

The function of the ACCEPT statement is to obtain low volume data from the device assigned as the system input device (SYSIN) or from a display station or SSP-ICF session. ACCEPT statement execution causes the transfer of data into the specified identifier. There is no editing or error checking of the incoming data. The formats of the ACCEPT statement are as follows:

Format 1

```
ACCEPT identifier [FROM mnemonic-name]
```

Format 2

```
ACCEPT identifier FROM {DATE}
                       {DAY }
                       {TIME}
```

Format 3

```
ACCEPT identifier-1 FROM mnemonic-name
[FOR {identifier-2}]
     {literal     }
```

Format 1 Considerations

This format is used to transfer data from an input device to the identifier. The identifier may be a group item, an elementary alphabetic or alphanumeric item, or a numeric data item with USAGE DISPLAY or USAGE COMPUTATIONAL.

If the FROM option is omitted, the system input device (requesting display station or invoking procedure) is assumed. If the program is invoked by a procedure, a record is read from the procedure for each ACCEPT statement until a /* is encountered. If the records in the procedure are exhausted or the program is not invoked by a procedure, the requesting display station is used. When the FROM option is specified, mnemonic-name must be associated with an input/output device that is specified in the SPECIAL-NAMES paragraph. The input/output device can be the display station console (REQUESTOR) or the system operator's console (SYSTEM-CONSOLE). If mnemonic-name is REQUESTOR and the job is entered by way of the JOBQ Command, the system operator's console is used.

When the device is the system input device, the following rules apply:

- An input record size of 120 characters is assumed.
- If identifier is longer than 120 characters, characters beyond the length of identifier are truncated.
- If identifier is less than 120 characters long, succeeding input records are read until the storage area of identifier is filled. If identifier is not an exact multiple of 120 characters, that part of the last input record that does not fit into identifier is truncated.

When the device is the display station keyboard, the same rules apply as when the device is the system input device except that the size is 60 characters.

The source of input data is dependent upon the type of program initiation as follows:

Method of Program Initiation	Mnemonic-name Associated with SYSTEM-CONSOLE	Mnemonic-name Associated with REQUESTOR	Data Source when FROM Option Omitted
JOBQ	System Console	System console	Data from next record in the procedure. If there is no data in the procedure, the input comes from the system console.
SRT	System Console	Display Station	Display Station
MRT	System Console	System Console	Can produce undesirable results. Specify the FROM option.

Input from the device can be terminated by entering a record beginning with /*. The /* is moved into the ACCEPT identifier with blank padding or truncation on the right. Any subsequent attempt to ACCEPT from the device is in error and execution will terminate. If the identifier is longer than the device size and the /* is entered for a succeeding input record, the identifier is padded to the right with blanks and the /* is treated as input to the next ACCEPT from the device.

Format 2 Considerations

This format is used to transfer the system information (program date and system time) to the identifier, using the rules for the MOVE statement without the CORRESPONDING option. Identifier can be a group item, or an elementary alphanumeric, alphanumeric edited, zoned decimal, packed decimal, binary, or numeric edited item.

DATE, DAY, and TIME implicitly have USAGE DISPLAY. DATE has the implicit PICTURE 9(6). The sequence of data elements from left to right is: two digits for year of century, two digits for month of year, two digits for day of month. Thus July 4, 1976 is expressed as 760704. DAY has the implicit PICTURE 9(5). The sequence of data elements from left to right is: two digits for year of century, three digits for day of year. Thus, July 4, 1976 is expressed as 76186. TIME has the implicit PICTURE 9(8). The sequence of data elements from left to right is: two digits for hour of day, two digits for minute of hour, two digits for second of minute, two digits for hundredths of second. Thus, 2:41 p.m. is expressed as 14410000. The time returned is the time when the ACCEPT statement executed.

Note: Time is always rounded up to the nearest second; therefore, a hundredths of a second is always expressed as 00.

The date is the last date specified in OCL for this job stream, or the current program date if no date has been specified in OCL since sign-on. If the program is an MRT, the data is the system date as of job initiation unless a different date is explicitly specified in the OCL for this job stream.

IBM Extension:

Format 3 Considerations

This format transfers data from the local data area or from the attribute record to identifier-1.

If the mnemonic-name is associated with LOCAL-DATA, the 256-byte local data area associated with the requestor terminal is moved into identifier-1.

If mnemonic-name is associated with ATTRIBUTE-DATA, identifier-1 must describe an attribute data record. (Atrribute data records are described under SPECIAL-NAMES paragraph in Chapter 7.) The attributes of the specified symbolic ID are moved into identifier-1. The TRANSACTION file must be open for this request.

The move into identifier-1 for both LOCAL-DATA and ATTRIBUTE-DATA takes place according to the rules for the MOVE statement for an alphanumeric group move without the CORRESPONDING option.

The FOR option is allowed only when mnemonic-name is associated with either ATTRIBUTE-DATA or LOCAL-DATA. Literal or the contents of identifier-2 is the symbolic ID of the display station or SSP-ICF session for which data is retrieved. A symbolic ID of blanks (or none specified) retrieves the attributes or local data from the requestor for which an input/output operation was most recently performed. In a program that has no TRANSACTION file, the local data is retrieved from the requestor for SRT batch jobs. The ID must be a two-character alphanumeric data item or literal.

Note: If the program is an MRT program, there is a local data area for each requestor and an additional local data area for the program. Prior to the successful completion of the first requestor's first input/output operation, this MRT local data area can be accessed. If no TRANSACTION file was specified, a symbolic ID of blanks returns the MRT's local data area.

If the mnemonic-name is associated with either SYSTEM-CONSOLE or REQUESTOR, the FOR option is not valid.

ACQUIRE Statement

The ACQUIRE statement attaches a display station or SSP-ICF session to the TRANSACTION file.

Format

The value of literal or identifier specifies the symbolic identification of a display station or SSP-ICF session that is to be associated with file-name. In order to be acquired, a display station must be in stand-by mode. In order to acquire an SSP-ICF session, it must be specified in the OCL SESSION statement for the job step.

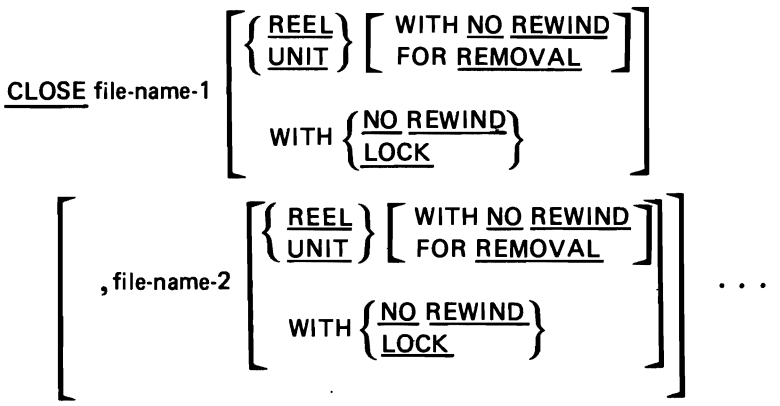
If literal is specified, it must be a two-character alphanumeric literal. If identifier is specified, it must refer to a two-character alphanumeric data item.

File-name must refer to a file whose organization is TRANSACTION.

CLOSE Statement

The CLOSE statement terminates the processing of files with optional lock.

Format



Each file-name designates a file upon which the CLOSE statement is to operate. The files need not have the same organization or access and must not be sort or merge files.

A CLOSE statement can be executed only for a file in an open mode. After successful execution of a CLOSE statement, the record area associated with the file-name is no longer available. Unsuccessful execution of a CLOSE statement leaves availability of the record data undefined.

After a CLOSE statement is successfully executed for the file, an OPEN statement for the file must be executed before any other input/output statement (except a SORT/MERGE statement with the USING or GIVING option) can refer explicitly or implicitly to the file. If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the CLOSE statement is executed. If the file is in an open status and the execution of a CLOSE statement is unsuccessful, the EXCEPTION/ERROR procedure (if specified) for this file is executed. If a CLOSE statement is not executed for an open file before a STOP RUN statement for this program is executed, results are unpredictable.

Specification of the lock option ensures that the file cannot be opened again in the program.

The REEL/UNIT option, the FOR REMOVAL option, and the NO REWIND option are treated as comments.

For special considerations concerning spooled printer files, see *Files* in Chapter 8.

IBM Extension:

TRANSACTION File Considerations

If a CLOSE statement is executed for the TRANSACTION file, no other statements that reference that file can be executed. A TRANSACTION file is locked when closed, whether or not the WITH LOCK option is specified.

Programming Note: For TRANSACTION files, the WITH LOCK option of the CLOSE statement should be specified for documentation.

DELETE Statement

The DELETE statement logically removes a record from an indexed or relative file. The DELETE statement can be successfully executed only on a system configured with extended data management.

Format

DELETE file-name RECORD [INVALID KEY imperative-statement]

When the DELETE statement is executed, the associated file must be opened in I-O mode. The file also must be created as delete-capable. This is done by specifying DFILE-YES when the file is created. (For more information on creating delete-capable files, see *FILE Statement* in the *SSP Reference Manual*.) File-name must be defined in an FD entry in the Data Division and must be the name of an indexed or relative file. After successful execution of a DELETE statement, the record is logically removed from the file and can no longer be accessed. For indexed files, the space that the record occupied cannot be used until the file is copied or reorganized. Execution of the DELETE statement does not affect the contents of the record area associated with file-name.

If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the DELETE statement is executed.

Sequential Access Mode

For a file in sequential access mode, the last prior input/output statement must be a successfully executed READ statement. When the DELETE statement is executed, the system logically removes the record retrieved by that READ statement. The current record pointer is not affected by execution of the DELETE statement.

The INVALID KEY option must not be specified for a file in sequential access mode. An EXCEPTION/ERROR procedure may be specified.

Random or Dynamic Access Mode

In random or dynamic access mode, DELETE statement execution results depend on whether the file organization is indexed or relative.

Indexed Files: When the DELETE statement is executed in random or dynamic access mode, the system logically removes the record identified by the contents of the RECORD KEY data item. If the file does not contain such a record, an INVALID KEY condition exists.

Relative Files: When the DELETE statement is executed in random or dynamic access mode, the system logically removes the record identified by the contents of the RELATIVE KEY data item. If the file does not contain such a record, an INVALID KEY condition exists.

Programming Notes

The DELETE statement logically removes the record from the file. For relative files, the space is then available for a new record with the same RELATIVE KEY value. For indexed files, a new record with the same RECORD KEY value can then be added. This record is not written in the space vacated by the deleted record. The space vacated by the deleted record is unavailable until the file is copied or reorganized.

DISPLAY Statement

The DISPLAY statement transfers low-volume data to an output device.

Format 1

```
DISPLAY { identifier-1 } [ , identifier-2 ] ... [ UPON mnemonic-name ]
        { literal-1    } [ , literal-2    ]
```

Format 2

```
DISPLAY { identifier-1 } [ , identifier-2 ] ... UPON mnemonic-name
        { literal-1    } [ , literal-2    ]
      [ FOR { literal-3    } ]
            { identifier-3 }
```

Format 1 Considerations

The DISPLAY statement transfers the contents of each operand to the output device in the left-to-right order in which the operands are listed. When a DISPLAY statement is executed, the data contained in the sending field is transferred to the output device. The size of the sending field is the total character count of all operands listed. If the total character count is less than the device maximum character count, the remaining rightmost characters are padded with spaces. If the total character count exceeds the maximum, as many records are written as are needed to display all operands. Any operand being printed when the end of a record is reached is continued in the next record.

IBM Extension: Identifiers described as USAGE COMPUTATIONAL-3 or USAGE COMPUTATIONAL-4 are converted to zoned decimal. No other items require conversion. Signed noninteger numeric literals are allowed.

Signed values in numeric fields cause the last character to show both the sign and number. For example, if SIGN WITH SEPARATE CHARACTER is not specified and two numeric items have the values -34 and 34, they are displayed as 3M and 34, respectively. If SIGN WITH SEPARATE CHARACTER is specified, a + or a - sign is displayed as either leading or trailing, depending on how the number was specified. If a figurative constant is specified as one of the operands, only a single occurence of the figurative constant is displayed.

If the UPON option is omitted, data is written to the current SYSLIST device. When the UPON option is specified, mnemonic-name must be associated in the SPECIAL-NAMES paragraph with either the display station console (REQUESTOR) or the system operator's console (SYSTEM-CONSOLE). The maximum logical record size is assumed for each device as follows:

Device	Maximum Logical Record Size
SYSLIST	120 characters
Display station	75 characters
System console	75 characters

The location of the output data is dependent upon the type of program initiation as follows:

Method of Initiation	Mnemonic-name Associated with SYSTEM-CONSOLE	Mnemonic-name Associated with REQUESTOR	UPON Option Omitted
JOBQ	System console	System console	Current SYSLIST device
SRT	System console	Display station	Current SYSLIST device
MRT	System console	System console	Current SYSLIST device

IBM Extension:

Format 2 Considerations

This format of the DISPLAY statement is applicable when mnemonic-name is associated with the system name LOCAL-DATA. For a description of the LOCAL-DATA area, see the LOCAL statement in the chapter on OCL statements in the System Support Reference Manual.

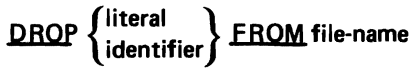
Literal-1 or the content of identifier-1 is written to the 256-byte local data area associated with the requestor.

Literal-3 or the contents of identifier-3 must be the valid symbolic ID of an attached requestor. Identifier-3 must be a two-character alphanumeric data item; literal-3 must be a two-character nonnumeric literal.

DROP Statement

The DROP statement releases a display station or SSP-ICF session from its association with the TRANSACTION file.

Format



The value of literal or identifier specifies the symbolic identification of the attached display station or SSP-ICF session that is to be released.

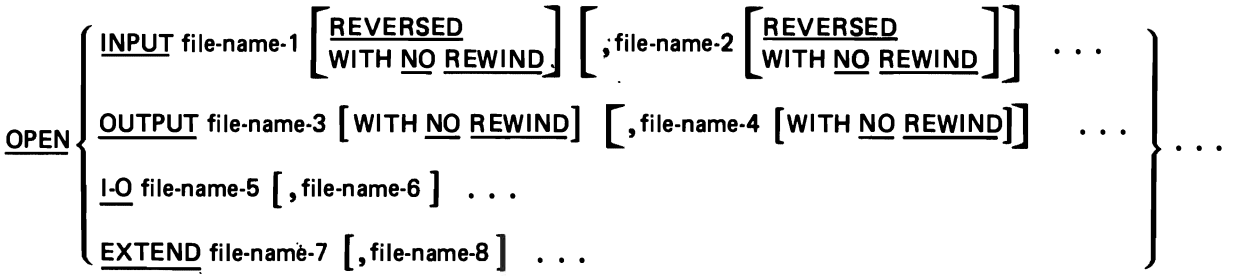
If literal is specified, it must be a two-character alphanumeric literal. If identifier is specified, it must refer to a two-character alphanumeric data item.

The DROP statement can only be used with a TRANSACTION file. At the end of program execution, all attached display stations and SSP-ICF sessions are implicitly released.

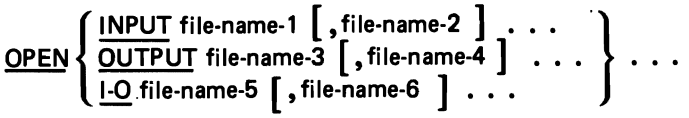
OPEN Statement

The OPEN statement initiates the processing of files. It also performs checking and/or writing of labels, and other input/output operations. The format of the OPEN statement is as follows:

Format 1–Sequential Files



Format 2–Indexed and Relative Files



Format 3–Transaction Files

```
OPEN I-O file-name-1
```

Each file-name designates a file upon which the OPEN statement is to operate. The files specified need not have the same organization or access. Each file-name must be defined in an FD entry in the Data Division, and must not name a sort or merge file. The FD entry must be equivalent to the information supplied when the file was defined.

The successful execution of an OPEN statement determines the availability of the file and results in that file being in the open mode. Before successful execution of the OPEN statement for a given file, no statement, except for a SORT or MERGE statement with the USING or GIVING option, that refers explicitly or implicitly to that file can be executed. The successful execution of the OPEN statement makes the associated record area available to the program; it does not obtain or release the first data record.

At least one of the options (INPUT, OUTPUT, I-O, or EXTEND) must be specified. More than one file-name may be specified in each option. The INPUT, OUTPUT, I-O, or EXTEND options may appear in any order.

The INPUT option permits opening the file for input operations. The I-O option permits opening the file for both input and output operations. The I-O option may be specified only for mass storage or TRANSACTION files. The INPUT and I-O options must not be specified when the file has not been already created.

The OUTPUT option permits opening the file for output operations. This option can only be specified when the file is being created. The OUTPUT option must not be specified for a file that contains records, or that did contain records that have been deleted.

Programming Note: The FILE OCL statement for an output file must contain a DISP-NEW parameter for proper processing.

The EXTEND option is valid only for sequential files and permits opening the file for output operations. It is discussed in the following section on Sequential Files.

A file may be opened for INPUT, OUTPUT, I-O, or EXTEND in the same program. After the first OPEN statement execution for a given file, each subsequent OPEN statement execution must be preceded by a successful CLOSE file statement execution without the LOCK option.

If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the OPEN statement is executed.

The REVERSED option is treated as a comment.

The NO REWIND option is treated as a comment.

Format 1—Sequential Files

The EXTEND option permits opening the file for output operations. When an OPEN EXTEND statement is executed, the file is prepared for the addition of records immediately following the last record in the file. Subsequent WRITE statements add records as if the file had been opened in OUTPUT mode. The EXTEND option can be specified when a file is being created. It can also be specified for a file that contains records, or that did contain records that have been deleted.

The EXTEND option has no meaning for a printer file, and it is ignored.

Execution of an OPEN INPUT or OPEN I-O statement sets the current record pointer to the first record existing in the file. If no records exist in the file, the current record pointer is set so that execution of the first READ statement results in an AT END condition.

For an input file, if SELECT OPTIONAL is specified in the file-control entry, OPEN statement execution causes the object program to check for the presence or absence of this file. If the file is absent, the first READ statement for this file causes the AT END condition to occur.

For special considerations concerning spooled printer files, see *Files* in Chapter 8.

Format 2—Indexed and Relative Files

Execution of an OPEN INPUT or OPEN I-O statement sets the current record pointer to the first record existing in the file; the record with the lowest record key value (indexed file) or lowest relative record number (relative file) is considered to be the first record in the file. If no records exist in the file, the current record pointer is set so that the first Format 1 READ statement executed results in an AT END condition.

IBM Extension:

Format 3-TRANSACTION Files

A TRANSACTION file must be opened with the I-O phrase.

A TRANSACTION file can be opened only once in a program.

READ Statement

The READ statement makes a record available to the object program before execution of any statement following the READ statement.

For sequential access, the READ statement makes available the next logical record from a disk file. For random access, the READ statement makes available a specified record from a disk file. When the READ statement is executed, the associated file must be opened in the INPUT or I-O mode. The formats of the READ statement are as follows:

Format 1–Sequential Access (Sequential Files)

```
READ file-name RECORD [INTO identifier] [AT END imperative-statement]
```

Format 2–Sequential Access (Relative and Indexed Files)

```
READ file-name [NEXT] RECORD [INTO identifier]
    [AT END imperative-statement]
```

Format 3–Random Access (Relative Files)

```
READ file-name RECORD [INTO identifier] [INVALID KEY imperative-statement]
```

Format 4–Random Access (Indexed Files)

```
READ file-name RECORD [INTO identifier]
    [KEY IS data-name]
    [INVALID KEY imperative-statement]
```

Format 5–Sequential Access (TRANSACTION File)

```
READ file-name RECORD
    [INTO identifier-1] [TERMINAL IS {identifier-2 / literal-1}]
    [NO DATA imperative-statement-1]
    [AT END imperative-statement-2]
```

File-name must be defined in a Data Division FD entry, and must not name a sort or merge file. If more than one record description entry is associated with file-name, these records automatically share the same storage area; that is, they are implicitly redefined. Before a READ statement is executed, the storage area is filled with blanks.

After a READ statement is executed, only those data items within the range of the current record are replaced; data items stored beyond that range are blanks. Figure 5-15 illustrates this concept. If no data items are defined, the entire record will be blank.

The FD entry for a TRANSACTION file is:

```
FD   INPUT-FILE LABEL RECORDS OMITTED.

01   RECORD-1  PICTURE X(30).

01   RECORD-2  PICTURE X(20).
```

After RECORD-1 is read, the input area contains:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ1234
```

If RECORD-2 consists of:

```
01234567890123456789
```

After RECORD-2 is read, the input area contains:

```
01234567890123456789␣␣␣␣␣␣␣␣␣␣
```

(Characters in the input area following RECORD-2 are blank.)

Figure 5-15. READ Statement with Multiple Record Descriptions

The AT END or INVALID KEY option must be specified if no implicit or explicit EXCEPTION/ERROR procedure is specified for this file.

If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the READ statement is executed.

Following unsuccessful READ statement execution, the contents of the associated record area and the position of the current record pointer are undefined.

INTO Identifier Option: The INTO identifier option makes a READ statement equivalent to:

```
READ file-name RECORD.

MOVE record-name TO identifier.
```

After successful execution of the READ statement, the current record becomes available both in the record-name and identifier.

When the INTO identifier option is specified, the current record is moved from the input area to the identifier area according to the rules for the MOVE statement without the CORRESPONDING option. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is transferred to identifier.

The INTO identifier option must not be specified when the file contains records of various sizes, as indicated by their record descriptions.

Sequential Access—Format 1 and Format 2

Formats 1 and 2 must be used for all files in sequential access mode. Execution of a Format 2 READ statement makes available the next logical record from the file. The record that is considered next depends upon the file organization.

NEXT RECORD Option: The next record is the succeeding logical record in key sequence. For indexed files, the key sequence is the ascending values of the current key of reference. For relative files, the key sequence is the ascending values of relative record numbers for records that exist in the file.

Before the READ statement is executed, the current record pointer must be set by a successful OPEN, START, or READ statement. When the READ statement is executed, the record indicated by the current record pointer is made available, if it is still accessible through the path indicated by the current record pointer. If the record is no longer accessible (for example, as a result of deletion of the record), the current record pointer is updated to indicate the next existing record in the file, and that record is made available.

For files in sequential access mode, the NEXT option can, but need not, be specified.

If the RELATIVE KEY clause is specified for sequentially accessed relative files, READ statement execution updates the RELATIVE KEY data item to indicate the relative record number of the record being made available.

AT END Condition: If no next logical record exists in the file when the READ statement is executed, an AT END condition occurs, and READ statement execution is unsuccessful. The following actions are taken, in the following order:

1. If the FILE STATUS clause is specified, the status key is updated to indicate an AT END condition.

2. If the AT END option is specified, control is transferred to the AT END imperative-statement. Any EXCEPTION/ERROR procedure for this file is not executed.

3. If the AT END option is not specified, then any EXCEPTION/ERROR procedure for this file is executed.

4. If neither the AT END nor the USE option is specified, a diagnostic message is issued.

When the AT END condition is recognized, a READ statement for this file must not be executed without first executing a successful CLOSE statement followed by a successful OPEN statement for this file.

When the AT END condition is recognized, a sequential access READ statement for this file must not be executed without first executing one of the following:

- 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

Random Access—Format 3 and Format 4

Format 3 or 4 must be specified for indexed and relative files in random access mode and also for files in the dynamic access mode when record retrieval is random.

Execution of the READ statement depends on the file organization as explained in following sections.

Files with Relative Organization: Execution of a Format 3 READ statement sets the current record pointer to the record whose relative record number is contained in the RELATIVE KEY data item and makes that record available. If the file does not contain such a record, the INVALID KEY condition exists, and READ statement execution is unsuccessful. The KEY option must not be specified for relative files.

Files with Indexed Organization: Execution of a Format 4 READ statement causes the value of the key of reference to be compared with the value of the corresponding key data item in the file records until the first record having an equal value is found. The current record pointer is positioned to this record, which is then made available. If no record can be identified, an INVALID KEY condition exists, and READ statement execution is unsuccessful.

If the KEY phrase is specified on a Format 4 READ statement, the statement is flagged as unsupported. No code is generated for the phrase. The System/34 does not support multiple keys for an indexed file. Therefore, specification of the key option is redundant, because the index file key must be specified with the RECORD KEY clause in the file-control entry in the Environment Division.

The RECORD KEY is the key of reference for a request. When dynamic access is specified, the RECORD KEY is also used as the key of reference for subsequent executions of sequential READ statements until a different key of reference is established.

Dynamic Access

For files with indexed or relative organization, dynamic access mode may be specified in the file-control entry. In dynamic access mode, either sequential or random record retrieval can be specified, depending on the format used.

If no more logical records exist in the file when the READ statement is executed, an AT END condition occurs. The same actions are taken as for files with sequential organization.

Format 2 with the NEXT option must be specified for sequential retrieval. All other rules for sequential access apply.

Format 3 or 4 must be specified for random retrieval. All other rules for random access apply.

Each successful sequential or random READ updates the current record pointer to the next logical record.

When DYNAMIC or SEQUENTIAL ACCESS to indexed files is specified, records added by a user program or IBM-supplied utility cannot be sequentially or randomly retrieved until a keysort has been performed by SSP, unless the (OCL) FILE statement contains an IFILE-YES parameter. For information concerning when keysorts are performed, see *Key Sorting for Indexed Files* in the *Concepts and Design Guide.*

IBM Extension:

TRANSACTION Files-Format 5

Format 5 must be used for the TRANSACTION file. Execution of the Format 5 READ statement makes a record available from the TRANSACTION file.

The TRANSACTION file must be open in the I-O mode at the time the READ statement is executed.

Upon successful execution of the READ statement, the terminal-id and function key fields of the CONTROL-AREA, if present, are filled in.

TERMINAL Option

The record to be made available by a READ statement is determined as follows:

- If the TERMINAL option is specified, the data record is made available from literal-1 or the contents of identifier-2 when identifier-2 contains a value other than blanks. Literal-1 or the nonblank contents of identifier-2 must be the symbolic ID of an attached display station or SSP-ICF session. Identifier-1 must be two-character alphanumeric; literal-1 must be two-character nonnumeric. When either literal-1 or the contents of identifier-2 are blank, the READ statement executes as though the TERMINAL option were omitted.

- If the TERMINAL option is omitted, the defaults are:
  - If a single display station or SSP-ICF session is attached to the file, the default is that display station or SSP-ICF session.
  - If multiple display stations and/or SSP-ICF sessions are attached to the file, there is no default. The data record made available is the first record input from any attached display station or SSP-ICF session.

Programming Note: Use of the TERMINAL option forces the next input to come from the specified display station or SSP-ICF session, unless literal-1 or identifier-2 contain blanks.

NO DATA Option

When the NO DATA option is specified, the imperative-statement specified is executed if a record cannot immediately be made available at the time of execution of the READ statement. After the imperative-statement is executed, the next sequential statement is executed.

When the NO DATA option is not specified, execution is suspended until a record becomes available.

AT END Condition

The AT END condition occurs when there are no attached display stations or SSP-ICF sessions for which an input operation is currently invited and the program is not a NEP. The AT END condition occurs for a NEP when there are no attached display stations or SSP-ICF sessions and the system operator has entered a STOP SYSTEM command.

Input is implicitly invited with each WRITE statement but can be suppressed by an option on the SFGR format or selected SSP-ICF predefined formats. When AT END condition occurs, the READ statement is unsuccessful and imperative-statement-2 is executed.

REWRITE Statement

The REWRITE statement logically replaces an existing record in a disk file. When the REWRITE statement is executed, the associated disk file must be opened in I-O mode.

Format

REWRITE record-name [FROM identifier] [INVALID KEY imperative-statement]

Record-name must be the name of a logical record in the File Section of the Data Division. Record-name must not be associated with a sort or merge file. Record-name may be qualified; it must not be subscripted or indexed. The number of character positions in record-name must equal the number of character positions in the record being replaced.

REWRITE statement execution replaces an existing record in the file with the information contained in record-name.

After successful execution of a REWRITE statement, the logical record is no longer available in record-name unless the associated file is named in a SAME RECORD AREA clause (in which case the record is also available as a record of the other files named in the SAME RECORD AREA clause).

The current record pointer is not affected by execution of the REWRITE statement.

If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the REWRITE statement is executed.

For files accessed sequentially, the last input/output statement successfully executed for the file must be a READ statement. When the REWRITE statement is executed, the record retrieved by that READ statement is logically replaced.

The FROM identifier option makes a REWRITE statement equivalent to:

```
MOVE identifier TO record-name

REWRITE record-name
```

After successful execution of the REWRITE statement, the current record may no longer be available in record-name, but is still available in identifier.

Sequential Files

The INVALID KEY option must not be specified for a file with sequential organization. An EXCEPTION/ERROR procedure may be specified.

Indexed Files

The record to be replaced is specified by the value contained in the RECORD KEY. When the REWRITE statement is executed for an indexed file that is accessed sequentially, the value specified in the RECORD KEY clause for the REWRITE statement must equal the value of the RECORD KEY data item in the last record read from the file. If the file is accessed randomly or dynamically, any record referenced by the RECORD KEY clause is rewritten.

An INVALID KEY condition exists when the access mode is sequential, and the value contained in the RECORD KEY of the record to be replaced does not equal the RECORD KEY data item of the last-retrieved record from the file.

If this condition exists, the INVALID KEY imperative-statement is executed, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, and the data in record-name is unaffected.

Relative Files

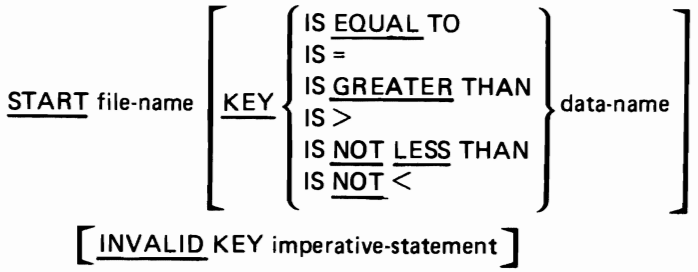
For relative files in the sequential access mode, the INVALID KEY option must not be specifed. An EXCEPTION/ERROR procedure may be specified.

When the access mode is random or dynamic, the record to be replaced is specified in the RELATIVE KEY data item. If the file does not contain the record specified, an INVALID KEY condition exists, and, if specified, the INVALID KEY imperative-statement is executed. The updating operation does not take place, and the data in record-name is unaffected.

START Statement

The START statement provides a means of positioning within an indexed or relative file for subsequent sequential record retrieval. When the START statement is executed, the associated indexed or relative file must be opened in INPUT or I-O mode.

Format



File-name must name a file with sequential or dynamic access. File-name must be defined in an FD entry in the Data Division, and must not name a sort or merge file.

KEY Option

When the KEY option is not specified, the EQUAL TO relational operator is implied.

When the KEY option is specified, the comparison specified in the KEY relational operator is made between data-name and the corresponding key field associated with the file's records. Data-name may be qualified; it may not be subscripted or indexed.

When the START statement is executed, a comparison is made between the current value in the key data-name and the corresponding key field in the file's records. The current record pointer is positioned to the logical record in the file whose key field satisfies the comparison.

If the FILE STATUS clause is specfied in the file-control entry, the associated status key is updated when the START statement is executed.

INVALID KEY Option

If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists; the position of the current record pointer is undefined, and (if specified) the INVALID KEY imperative-statement is executed.

The INVALID KEY option must be specified if no EXCEPTION/ERROR procedure is explicitly or implicitly specified for this file.

Indexed Files

When the KEY option is not specified, the key data item used for the EQUAL TO comparison is the RECORD KEY. When START statement execution is successfully completed, the RECORD KEY becomes the key of reference for subsequent READ statements.

When the KEY option is specified, the key data item used for the comparison is data-name, which can be:

- The RECORD KEY
- An alphanumeric data item subordinate to a record key whose leftmost character position corresponds to the leftmost character position of that record key. This data item may be qualified.

The current record pointer is positioned to the first record in the file whose key field satisfies the comparison. If the operands in the comparison are of unequal length, the comparison proceeds as if the longer field were truncated on the right to the length of the shorter field. All other numeric and nonnumeric comparison rules apply except that the PROGRAM COLLATING SEQUENCE clause, if specified, has no effect.

When START statement execution is successful, the RECORD KEY with which the data-name is associated becomes the key of reference for subsequent READ statements.

When START statement execution is unsuccessful, the key of reference is undefined.

Relative Files

When the KEY option is specified, data-name must specify the RELATIVE KEY.

Whether or not the KEY option is specified, the key data item used in the comparison is the RELATIVE KEY data item. The current record pointer is positioned to the logical record in the file whose key satisfies the comparison.

WRITE Statement

The WRITE statement releases a logical record for an output or input/output file. A WRITE statement can be specified for:

- TRANSACTION files
- Relative files opened in OUTPUT or I-O mode
- Indexed files opened in OUTPUT or I-O mode
- Sequential files opened in OUTPUT or EXTEND mode

The formats of the WRITE statement are:

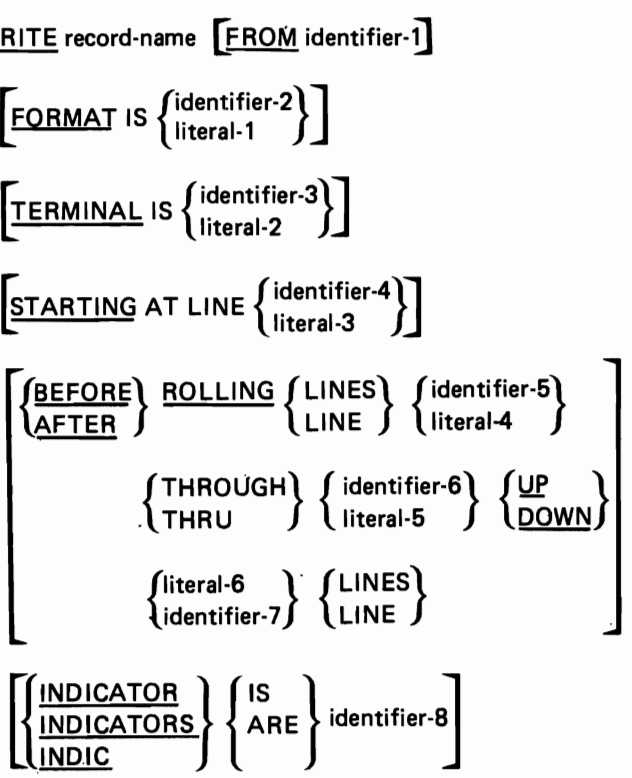
Format 1

```
WRITE record-name [ FROM identifier-1 ]
    [ { BEFORE } ADVANCING { { identifier-2 } [ LINE ] }  ]
    [ { AFTER  }           { { integer      } [ LINES ] }  ]
    [                      { { mnemonic-name }          }  ]
    [                      { { PAGE          }          }  ]
    [ AT { END-OF-PAGE } imperative-statement ]
         { EOP         }
```

Format 2

```
WRITE record-name [ FROM identifier ] [ INVALID KEY imperative-statement ]
```

Format 3–TRANSACTION File



This page is intentionally left blank.

Record-name must be the name of a logical record in the File Section of the Data Division. Record-name may be qualified. Record-name must not be associated with a sort or a merge file.

The maximum record size for the file is established at the time the file is created, and cannot subsequently be changed. User-defined record lengths that are not compatible with the record length specified in the file may result in a nonzero file-status at open time and the following results during output to the file:

- A user-defined length greater than file-specified length causes truncation. If the file is empty the larger record length is used.

- A user-defined length less than file-specified length causes padding with blanks.

Execution of the WRITE statement releases a logical record to the file associated with record-name. After the WRITE statement is executed, the logical record is no longer available in record-name, unless either of the following is true:

- The associated file is named in a SAME RECORD AREA clause. If so, the record is also available as a record of the other files named in the SAME RECORD AREA clause.

- The WRITE statement is unsuccessful due to a boundary violation (beyond extent).

If either condition is true, the logical record is still available in record-name.

The current record pointer is not affected by execution of the WRITE statement.

The number of character positions required to store the record in a file may or may not be the same as the number of character positions defined by the logical description of that record in the COBOL program. (See the descriptions of the PICTURE and USAGE clauses in Chapter 4.)

If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the WRITE statement is executed whether or not execution is successful.

When an attempt is made to write beyond the externally defined boundaries of the file, WRITE statement execution is unsuccessful, and an EXCEPTION/ERROR condition exists. The status key, if specified, is updated, and if an explicit or implicit EXCEPTION/ERROR procedure is specified for the file, the procedure is executed; if no such procedure is specified, the results are unpredictable.

FROM Identifier Option: The FROM identifier option makes a WRITE statement equivalent to:

```
MOVE identifier TO record-name

WRITE record-name
```

After successful execution of the WRITE or REWRITE statement, the current record may no longer be available in record-name, but is still available in identifier.

Format 1 Considerations

The ADVANCING and END-OF-PAGE options control the vertical positioning of each line on a printed page.

For the first WRITE statement to a printer file, the linage counter is set at 1. If the line is to be at the top of the page, use WRITE line AFTER ADVANCING 0 LINES.

ADVANCING Option: When the ADVANCING option is omitted, automatic line advancing is provided. The default statement is AFTER ADVANCING 1 LINE. When the ADVANCING option is specified, the following rules apply:

- When BEFORE ADVANCING is specified, the line is printed before the page is advanced.

- When AFTER ADVANCING is specified for the first WRITE, a blank page will be printed.

  If linage is not specified, the linage counter is undefined until the first WRITE statement. When AFTER ADVANCING is specified for the first WRITE, the linage counter is set at the top of the first page.

- When identifier-2 is specified, the page is advanced the number of lines equal to the current value in identifier-2. Identifier-2 must name an elementary integer data item. Identifier-2 may be zero.

- When integer is specified, the page is advanced the number of lines equal to the value of integer. Integer may be zero.

- When a mnemonic-name is specified, a page eject or space suppression takes place. The mnemonic-name must be equated with function-name-1 in the SPECIAL-NAMES paragraph. This option is not valid if a LINAGE clause is specified in the FD entry for this file.

- When PAGE is specified, the record is printed on the logical page BEFORE or AFTER (depending on the option used) the device is positioned to the next logical page. If PAGE has no meaning for the device used, then BEFORE or AFTER ADVANCING 1 LINE is provided depending on the option specified.

  If the FD entry contains a LINAGE clause, the repositioning is to the first printable line of the next page as specified in that clause. If the LINAGE clause is omitted, the repositioning is to line 1 of the next page.

If the LINAGE clause is specified for this file, the associated LINAGE-COUNTER special register is modified during the execution of the WRITE statement, according to the following rules:

- If ADVANCING PAGE is specified, LINAGE-COUNTER is reset to 1.

- If ADVANCING identifier-2 or integer is specified, LINAGE-COUNTER is incremented by the value in identifier-2 or integer.

- If the ADVANCING option is omitted, LINAGE-COUNTER is incremented by 1.

- When the device is repositioned to the first printable line of a new page, LINAGE-COUNTER is reset to 1.

END-OF-PAGE Option: The key words END-OF-PAGE and EOP are equivalent.

When the END-OF-PAGE option is specified, the FD entry for this file must contain a LINAGE clause. When END-OF-PAGE is specified, and the logical end of the printed page is reached during execution of the WRITE statement, the END-OF-PAGE imperative-statement is executed.

The logical end of the printed page is specified in the associated LINAGE clause.

An END-OF-PAGE condition is reached when execution of a WRITE END-OF-PAGE statement causes printing or spacing within the footing area of a page body. This occurs when execution of such a WRITE statement causes the value in the LINAGE-COUNTER to equal or exceed the value specified in the WITH FOOTING option of the LINAGE clause. The WRITE statement is executed and then the END-OF-PAGE imperative-statement is executed.

An automatic page overflow condition is reached whenever the execution of any given WRITE statement with or without the END-OF-PAGE option cannot be completely executed within the current page body. This occurs when a WRITE statement, if executed, would cause the value in the LINAGE-COUNTER to exceed the number of lines for the page body specified in the LINAGE clause. In this case, the line is printed BEFORE or AFTER the device is repositioned to the first printable line on the next logical page, as specified in the LINAGE clause. If the END-OF-PAGE option is specified, the END-OF-PAGE imperative-statement is then executed.

The END-OF-PAGE condition and automatic page overflow condition occur simultaneously when:

- The WITH FOOTING option of the LINAGE clause is not specified. This happens because there is no distinction between the END-OF-PAGE condition and the page overflow condition.

- The WITH FOOTING option is specified, but the execution of a WRITE statement would cause the LINAGE-COUNTER to exceed both the footing value and the page body value specified in the LINAGE clause.

Format 2 Considerations

This format is valid only for indexed and relative files.

Indexed Files: When the WRITE statement is executed, the system releases the record. Before the WRITE statement is executed, the user must set the record key (the RECORD KEY data item, as defined in the file-control entry) to the desired value. RECORD KEY values must be unique within a file.

When ACCESS IS SEQUENTIAL is specified in the file-control entry, records must be released in ascending order of RECORD KEY values.

When ACCESS IS RANDOM or ACCESS IS DYNAMIC is specified in the file-control entry, records can be released in any user-specified order. The WRITE statement cannot be used when ACCESS IS DYNAMIC is specified and the file is opened in I-O mode.

The *INVALID KEY Option* must be specified if an explicit or implicit EXCEPTION/ERROR procedure is not specified for this file.

When the INVALID KEY condition is recognized, WRITE statement execution is unsuccessful, and the contents of the record are unaffected. Program execution proceeds according to the rules for an INVALID KEY condition. An INVALID KEY condition is caused by any of the following:

- ACCESS SEQUENTIAL is specified, and the file is opened OUTPUT; and the value of the record key is not greater than that for the previous record.
- The file is opened I-O, and the value of the record key equals that of an already existing record.
- When an attempt is made to write beyond the externally defined boundaries of the file.

Note: The BYPASS-YES parameter on the FILE OCL statement allows the COBOL programmer to suppress duplicate key checking when adding a record to an indexed file. It is the programmer's responsibility to ensure that duplicate keys are not added. The BYPASS-YES parameter is not intended as support for duplicate keys. Specifying the BYPASS-YES parameter can improve system performance, but results in nonstandard COBOL file processing. For more information on the BYPASS-YES parameter, see the *FILE statement* in the *SSP Reference Manual.*

Relative Files: The WRITE statement is valid for both OUTPUT and I-O files.

For OUTPUT files, the WRITE statement causes the following actions:

- If ACCESS IS SEQUENTIAL is specified, the first record released has relative record number 1; the second, number 2; the third, number 3; and so on. If the RELATIVE KEY is specified in the file-control entry, the relative record number of the record just released is placed in the RELATIVE KEY during execution of the WRITE statement.
- If ACCESS IS RANDOM or ACCESS IS DYNAMIC is specified, the RELATIVE KEY must contain the desired relative record number for this record before the WRITE statement is issued. When the WRITE statement is executed, this record is placed at the specified relative record number position in the file if this relative record position is vacant.

For I-O files, when ACCESS IS RANDOM or ACCESS IS DYNAMIC is specified, new records are inserted into the files. The RELATIVE KEY must contain the desired relative record number for this record before the WRITE statement is issued. When the WRITE statement is executed, this record is placed at the specified relative record number position in the file.

The *INVALID KEY Option* must be specified if an explicit or implicit EXCEPTION/ERROR procedure is not specified for this file.

When the INVALID KEY condition is recognized, WRITE statement execution is unsuccessful, and the contents of the record area are unaffected. Program execution proceeds according to the rules for an INVALID KEY condition. An INVALID KEY condition is caused by either of the following:

- ACCESS IS RANDOM or ACCESS IS DYNAMIC is specified, and the RELATIVE KEY specifies a record that already contains data.
- An attempt is made to write beyond the externally defined boundaries of the file.

Note: This format of the WRITE statement can be successfully executed only on a system configured with extended disk data management. The files used must be created and prcessed as delete-capable files.

IBM Extension:

Format 3 Considerations

This format is valid only for the TRANSACTION file.

The WRITE statement releases a logical record to the TRANSACTION file. This file must be opened in the I-O mode at the time the WRITE statement is executed.

Literal-1 and literal-2 must be nonnumeric. Literal-3, literal-4, literal-5, and literal-6 must be numeric.

Identifier-2 must be an alphabetic or alphanumeric data item and identifier-3 must an an alphanumeric data item. Identifier-4, identifier-5, identifier-6, and identifier-7 must be elementary numeric items. Identifier-8 must be either an elementary Boolean data item specified without the OCCURS clause, or a group item that has Boolean data elementary items subordinate to it.

FORMAT Option

The record specified by the record-name is sent to the specified or implied destination using the named format. A format must be specified for the first WRITE verb executed. If subsequent WRITE operations do not include a FORMAT option, the most recently used format is used. The FORMAT option contains the name of the screen format used when data is written to the display station. This format must be in the format load member. The member name is specified as part of the assignment-name in the ASSIGN clause for the TRANSACTION file.

Writing to the Error Line: If the format name used for the write operation is the literal 'ERRLINE', System/34 COBOL generates a write to the error line of the display station instead of a write with format. A line written to the error line cannot exceed 78 characters in length. A write to the error line causes the last line of output on the screen to be saved, and the output record to replace the bottom line on the screen. When the operator presses the RESET key, the original line reappears.

Note: A WRITE statement that writes to the error line cannot specify ROLLING BEFORE or AFTER.

Interactive Communications Feature: Special format names are recognized by Data Management that provide the COBOL user SSP-ICF functions. The uses of these special format names and the functions of ICF are described in the ICF Reference Manual. The system defined special format names begin with two dollar signs ($$). You should not begin your display screen format names with $$.

TERMINAL Option

The TERMINAL phrase is used to specify the destination to which the record is to be sent. If the TERMINAL option is not specified for a single device file, that device is the destination. If the TERMINAL option is not specified for a multiple device file, the most recent source or destination identifier is used as the destination.

STARTING Option

The STARTING phrase contains the starting line number for screen formats that use the variable start line option. If the value of this element is less than 01, a value of 01 is assumed. The maximum value is one less than the size of the screen. If the screen format does not specify this option, Display Station Data Management (DSDM) ignores this value.

ROLLING Option

The ROLLING option allows you to move the data currently displayed on the display screen. All or part of the data on the screen can be rolled up or down. The lines vacated by the rolled data are cleared, and can have another screen format written into them.

Rolling is specified on the WRITE statement that is writing a new format to the display screen. The number of lines you want to roll, how many lines you want to roll these lines, and whether the roll operation is up or down must be specified.

Note: The value specified by identifier-5 (or literal-4) must be less than the value specified by identifier-6 (or literal-5).

Rolling ignores field attributes. The data is rolled exactly as it appears on the display screen. Its associated attributes (for example, whether it is an input field or an input/output field) are not rolled with the data and are lost. Therefore, after a field has been rolled, it can no longer be input capable.

For an example of using the ROLLING option, see WRITE Statement in Chapter 7.

INDICATOR Option

The INDICATOR phrase is used to specify the name of an area that contains SFGR indicator information. Display Station Data Management (DSDM) ignores provided indicators that are not specified on the SFGR format. Indicators not provided in the indicator area are considered by DSDM to be off.

Arithmetic statements are used for computations. Individual operations are specified by the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements. The COMPUTE statement may be used to symbolically combine these operations in a formula.

Arithmetic Statement Operands

The data description of operands in an arithmetic statement need not be the same. Throughout the calculation, the compiler supplies any necessary data conversion and decimal point alignment.

Size of Operands

The maximum size of each operand is 18 decimal digits. The composite of operands (a hypothetical data item resulting from the superposition of the operands aligned by decimal point) must not contain more than 18 decimal digits.

For the ADD and SUBTRACT statements, the composite of operands is determined by superimposing all operands in a given statement except those following the word GIVING.

For the MULTIPLY statement, the composite of operands is determined by superimposing all receiving data items.

For the DIVIDE statement, the composite of operands is determined by superimposing all receiving data items except the REMAINDER data item.

For the COMPUTE statement, the restriction on composite of operands does not apply.

For example, the items A, B, and C are defined in the Data Division as follows:

```
77 A PICTURE S9(7)V9(5).

77 B PICTURE S9(11)V99.

77 C PICTURE S9(12)V9(3).
```

If the statement ADD A, B TO C is executed, then the composite of operands for this statement consists of 17 decimal digits. It has the following implicit description:

```
Composite-of-Operands PICTURE S9(12)V9(5).
```

Overlapping Operands

When operands in an arithmetic statement share part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

Multiple Results

When an arithmetic statement has multiple results, execution conceptually proceeds as follows:

- The statement performs all arithmetic operations to find the result to be placed in the receiving items and stores that result in a temporary location.
- A sequence of statements transfers or combines the value of this temporary result with each single receiving field. The statements are considered to be written in the same left-to-right order that the multiple results are listed.

For example, executing the following statement:

```
ADD A, B, C TO C, D(C), E
```

is equivalent to executing the following series of statements:

```
ADD A, B, C GIVING TEMP

ADD TEMP TO C

ADD TEMP TO D(C)

ADD TEMP TO E
```

TEMP is a compiler-supplied temporary result field. When the addition operation for D(C) is performed, the subscript C contains the new value of C.

Programming Notes

In all arithmetic statements, it is the user's responsibility to define data with enough digits and decimal places to ensure accuracy in the final result.

Common Options

There are several options common to the arithmetic statements. They are the CORRESPONDING option, the GIVING Option, the ROUNDED option, and the SIZE ERROR option. Their description precedes the descriptions of the individual statements.

CORRESPONDING Option

The CORRESPONDING option allows operations to be performed on elementary items of the same name simply by specifying the group items to which they belong.

The CORRESPONDING option is valid in the ADD, SUBTRACT, and MOVE statements. The abbreviation CORR is equivalent to the key word CORRESPONDING.

Both identifiers following the key word CORRESPONDING must name group items. In this discussion, these identifiers are referred to as d1 and d2.

A pair of subordinate data items, one from d1 and one from d2, correspond if the following conditions are true:

- In an ADD or SUBTRACT statement, both of the subordinate items are elementary numeric data-items.
- In a MOVE statement, at least one of the subordinate items is elementary.
- The two subordinate items have the same name and the same qualifiers up to but not including d1 and d2.
- The subordinate items are not identified by the key word FILLER.

- The subordinate items do not include a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause in their descriptions; if such a subordinate item is a group, the items subordinate to it are also ignored. However, d1 and d2 themselves may contain or be subordinate to items containing a REDEFINES or OCCURS clause in their descriptions.

  For example, two data hierarchies are defined as follows:

```
05  ITEM-1 OCCURS 6 INDEXED BY X.
      10 ITEM-A ...
      10 ITEM-B ...
      10 ITEM-C REDEFINES ITEM-B ...
05  ITEM-2
      10 ITEM-A ...
      10 ITEM-B ...
      10 ITEM-C ...
```

  If ADD CORR ITEM-2 TO ITEM-1(X) is specified, ITEM-A and ITEM-A(X) and ITEM-B and ITEM-B(X) are considered to be corresponding and are added together. ITEM-C and ITEM-C(X) are not included because ITEM-C(X) includes a REDEFINES clause in its data description. ITEM-1 is valid as either d1 or d2.

- Neither d1 nor d2 is described as a level 66, 77 or 88 item, or as a FILLER or USAGE IS INDEX item.

GIVING Option

If the GIVING option is specified, the value of the identifier that follows the word GIVING is set equal to the calculated result of the arithmetic operation. Because this identifier is not involved in the computation, it may be a numeric edited item.

ROUNDED Option

After decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is compared with the number of places provided for the fraction of the resultant identifier.

If the size of the fractional result exceeds the number of places provided for its storage, truncation occurs unless the ROUNDED option is specified. When the ROUNDED option is specified, the least significant digit of the resultant identifier has its value increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

When the resultant identifier is described by a PICTURE clause containing rightmost Ps and when the number of places in the calculated result exceeds the number of integer positions specified, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

SIZE ERROR Option

A size error condition exists if, after decimal point alignment, the value of a result exceeds the largest value that can be contained in the resultant field. Division by zero and zero raised to the zero power always causes a size error condition.

In the ADD, SUBTRACT, and COMPUTE statements, the size error condition applies only to final results. In the MULTIPLY and DIVIDE statements, the size error condition applies both to final results and intermediate results.

If the ROUNDED option is specified, rounding takes place before size error checking.

When a size error occurs, the subsequent action of the program depends on whether or not the SIZE ERROR option is specified.

If the SIZE ERROR option is not specified and a size error condition occurs, the value of the affected resultant identifier is unpredictable. When multiple receivers are specified, those that do not have a size error are not affected by receivers that do have the error.

If the SIZE ERROR option is specified and a size error condition occurs, the error results are not placed in the receiving identifier. After completion of the execution of the arithmetic operation, the imperative-statement in the SIZE ERROR option is executed.

If an individual arithmetic operation causes a size error condition for ADD CORRESPONDING and SUBTRACT CORRESPONDING statements, the SIZE ERROR imperative-statement is not executed until all of the individual additions or subtractions have been completed.

ADD Statement

The ADD statement causes two or more numeric operands to be summed and the result to be stored. The formats of the ADD statement are as follows:

Format 1

```
ADD {identifier-1} [,identifier-2] ... TO identifier-m [ROUNDED]
    {literal-1   } [,literal-2   ]

    [,identifier-n [ROUNDED]] ... [ON SIZE ERROR imperative-statement]
```

Format 2

```
ADD {identifier-1} , {identifier-2} [,identifier-3] ...
    {literal-1   }   {literal-2   }  [,literal-3   ]

    GIVING identifier-m [ROUNDED] [,identifier-n [ROUNDED]] ...

    [ON SIZE ERROR imperative-statement]
```

Format 3

```
ADD {CORRESPONDING} identifier-1 TO identifier-2 [ROUNDED]
    {CORR         }

    [ON SIZE ERROR imperative-statement]
```

In Formats 1 and 2, each identifier, except those following the key word GIVING must name an elementary numeric item. In Format 2, each identifier following the key word GIVING must name an elementary numeric or numeric edited item. In Format 3, each identifier must name a group item. In all formats, each literal must be a numeric literal.

In Format 1, all identifiers or literals preceding the key word TO are added together, and this sum is added to and stored immediately in identifier-m. If specified, the sum is then added to and stored immediately in identifier-n, and so on.

In Format 2, at least two operands must precede the key word GIVING. The values of these operands are added together, and the sum is stored as the new value of identifier-m, and, if specified, identifier-n, and so on.

In Format 3, elementary data items within identifier-1 are added to and stored in the corresponding elementary items within identifier-2.

If the composite of the operands is 18 digits or less, the compiler ensures that enough places are carried so that no significant digits are lost during execution.

COMPUTE Statement

The COMPUTE statement assigns the value of an arithmetic expression to one or more data items.

Format

```
COMPUTE identifier-1 [ROUNDED] [ ,identifier-2 [ROUNDED] ] . . .

    = arithmetic-expression [ON SIZE ERROR imperative-statement]
```

The COMPUTE statement allows the user to combine arithmetic operations without the restrictions imposed by the rules for the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements on the composite of operands or on receiving data items. (For more information about the composite of operands, see "Size of Operands" under *Arithmetic Statements* in this chapter.)

The identifiers that appear to the left of the equal sign must name either elementary numeric items or elementary numeric edited items.

When the COMPUTE statement is executed, the value of the arithmetic expression is calculated; then this value is stored as the new value of identifier-1, identifier-2, and so on, in turn.

The arithmetic expression may be any meaningful combination of identifiers, numeric literals, and arithmetic operators.

An arithmetic expression consisting of a single identifier or literal allows the user to set identifier-1, and so on, equal to the value of that identifier or literal.

Programming Notes

- When arithmetic operations must be combined, the COMPUTE statement is more efficient than the separate arithmetic statements written in series.

- The limitation on intermediate result fields exists in the COMPUTE statement as well as in the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements. Refer to *Appendix D* for a description of intermediate-result algorithms.

DIVIDE Statement

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal to the quotient and remainder. The formats of the DIVIDE statement are:

Format 1

```
DIVIDE {identifier-1} INTO identifier-2 [ROUNDED]
       {literal-1   }
    [,identifier-3 [ROUNDED]] ... [ON SIZE ERROR imperative-statement]
```

Format 2

```
DIVIDE {identifier-1} {INTO} {identifier-2} GIVING identifier-3 [ROUNDED]
       {literal-1   } {BY  } {literal-2   }
    [,identifier-4 [ROUNDED]] ... [ON SIZE ERROR imperative-statement]
```

Format 3

```
DIVIDE {identifier-1} {INTO} {identifier-2} GIVING identifier-3 [ROUNDED]
       {literal-1   } {BY  } {literal-2   }
    REMAINDER identifier-4 [ON SIZE ERROR imperative-statement]
```

Each identifier except those following the key words GIVING and REMAINDER must name an elementary numeric item. Each identifier following the key words GIVING and REMAINDER must name an elementary numeric or numeric edited item. Each literal must be a numeric literal.

In Format 1, the value of literal-1 or identifier-1 is divided into the value of identifier-2; then the quotient is placed in identifier-2. If identifier-3 is specified, the value of literal-1 or identifier-1 is divided into identifier-3; then the quotient is placed in identifier-3, and so on.

In Format 2, the value of identifier-1 or literal-1 is divided into/by the value of identifier-2 or literal-2. The value of the quotient is stored in identifier-3, and (if specified) identifier-4, and so on.

In Format 3, the value of identifier-1 or literal-1 is divided into/by identifier-2 or literal-2. The value of the quotient is stored in identifier-3, and the value of the remainder is stored in identifier-4.

The remainder is defined as the result of subtracting the product of the quotient and the divisor from the dividend. If identifier-3 (the quotient) is a numeric edited field, the quotient used to calculate the remainder is an intermediate field that contains the unedited quotient.

In addition to the conditions for common options, the following considerations apply when the ROUNDED and SIZE ERROR options are used in format 3.

- When the ROUNDED option is specified the quotient used to calculate the remainder is an intermediate field which contains the quotient truncated rather than rounded.

- When the ON SIZE ERROR option is specified and the size error conditions occurs on the quotient, no remainder calculation is meaningful. Therefore, the contents of the quotient field (identifier-3) and the remainder field (identifier-4) are unchanged.

- When the ON SIZE ERROR option is specified and the size error occurs on the remainder, the contents of the remainder field (identifier-4) are unchanged.

Note: In the last two preceding cases, the user must analyze the results to determine which situation has actually occurred.

MULTIPLY Statement

The MULTIPLY statement causes numeric items to be multiplied and sets the values of data items equal to the results. The formats of the MULTIPLY statement are:

Format 1

```
MULTIPLY { identifier-1 } BY identifier-2 [ ROUNDED ]
         { literal-1    }

    [ , identifier-3 [ ROUNDED ] ]  . . .  [ ON SIZE ERROR imperative-statement ]
```

Format 2

```
MULTIPLY { identifier-1 } BY { identifier-2 } GIVING identifier-3 [ ROUNDED ]
         { literal-1    }    { literal-2    }

    [ , identifier-4 [ ROUNDED ] ]  . . .  [ ON SIZE ERROR imperative-statement ]
```

Each identifier except those following the key word GIVING must name an elementary numeric item. Each identifier following the key word GIVING must name an elementary numeric or numeric edited item. Each literal must be a numeric literal.

In Format 1, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2; the product is then placed in identifier-2. If identifier-3 is specified, the value of identifier-1 or literal-1 is multiplied by the value of identifier-3; the product is then placed in identifier-3, and so on.

In Format 2, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2 or literal-2; the product is then stored in identifier-3, and, if specified, identifier-4, and so on.

SUBTRACT Statement

The SUBTRACT statement causes either one, or the sum of two or more numeric items to be subtracted from one or more numeric items and the result to be stored. The formats of the SUBTRACT statement are:

Format 1

```
SUBTRACT { identifier-1 } [ ,identifier-2 ] ... FROM identifier-3 [ ROUNDED ]
         { literal-1    } [ ,literal-2    ]

    [ ,identifier-4 [ ROUNDED ] ] ... [ ON SIZE ERROR imperative-statement ]
```

Format 2

```
SUBTRACT { identifier-1 } [ ,identifier-2 ] ... FROM { identifier-3 }
         { literal-1    } [ ,literal-2    ]        { literal-3    }

    GIVING identifier-4 [ ROUNDED ] [ ,identifier-5 [ ROUNDED ] ] ...

    [ ON SIZE ERROR imperative-statement ]
```

Format 3

```
SUBTRACT { CORRESPONDING } identifier-1 FROM identifier-2 [ ROUNDED ]
         { CORR          }

    [ ON SIZE ERROR imperative-statement ]
```

In Formats 1 and 2, each identifier except those following the key word GIVING must name an elementary numeric item. In Format 2, each identifier following the key word GIVING must name a numeric elementary or numeric edited elementary item. In Format 3, each identifier must name a group item. In all formats, each literal must be a numeric literal.

In Format 1, all identifiers or literals preceding the key word FROM are added together, and this sum is subtracted from and stored immediately in identifier-3, and then, if specified, subtracted from and stored immediately in identifier-4, and so on.

In Format 2, all identifiers or literals preceding the key word FROM are added together and this sum is subtracted from identifier-3 or literal-3. The result of the subtraction is stored as the new value of identifier-4, and, if specified, identifier-5, and so on.

In Format 3, elementary data items within identifier-1 are subtracted from and stored in the corresponding elementary data items within identifier-2.

If the composite of the operands is 18 digits or less, the compiler ensures that enough places are carried so that no significant digits are lost during execution.

DATA MANIPULATION STATEMENTS

Movement and inspection of data are the functions of the following COBOL statements: INSPECT, MOVE, STRING, and UNSTRING.

When the sending and receiving fields of a data manipulation statement share a part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

INSPECT Statement

The INSPECT statement specifies that characters in a data item are to be counted, replaced, or counted and replaced. The formats of the INSPECT statement are:

Format 1

```
INSPECT identifier-1 TALLYING
    { ,identifier-2 FOR { ,{ {ALL|LEADING} {identifier-3|literal-1} | CHARACTERS } }
      [ {BEFORE|AFTER} INITIAL {identifier-4|literal-2} ] } ...  } ...
```

Format 2

```
INSPECT identifier-1 REPLACING
    { CHARACTERS BY {identifier-6|literal-4} [ {BEFORE|AFTER} INITIAL {identifier-7|literal-5} ]
    { ,{ALL|LEADING|FIRST} { ,{identifier-5|literal-3} BY {identifier-6|literal-4}
      [ {BEFORE|AFTER} INITIAL {identifier-7|literal-5} ] } ...  } ...
```

Format 3

```
INSPECT identifier-1 TALLYING
    { ,identifier-2 FOR { ,{ {ALL|LEADING} {identifier-3|literal-1} | CHARACTERS } }
      [ {BEFORE|AFTER} INITIAL {identifier-4|literal-2} ] } ...  } ...
  REPLACING
    { CHARACTERS BY {identifier-6|literal-4} [ {BEFORE|AFTER} INITIAL {identifier-7|literal-5} ]
    { ,{ALL|LEADING|FIRST} { ,{identifier-5|literal-3} BY {identifier-6|literal-4}
      [ {BEFORE|AFTER} INITIAL {identifier-7|literal-5} ] } ...  } ...
```

Either the TALLYING or the REPLACING option must be specified. Both the TALLYING and REPLACING options may be specified. If both TALLYING and REPLACING are specified (Format 3), all tallying is performed before any replacement is made.

Identifier-1 is the inspected item. Identifier-1 must be an elementary or group item with USAGE DISPLAY.

All other identifiers except identifier-2 (the count field) must be elementary alphabetic, alphanumeric, or zoned decimal items. Each is treated according to its data category. Each data category is treated as follows:

- Alphabetic or alphanumeric items are treated as a character-string.
- Alphanumeric edited, numeric edited, or unsigned numeric (zoned decimal) items are treated as though redefined as alphanumeric and the INSPECT statement refers to the alphanumeric item.
- Signed numeric (zoned decimal) items are treated as though moved to an unsigned zoned decimal item of the same length, and then treated as though redefined as alphanumeric. The INSPECT statement refers to the alphanumeric item.

Each literal must be nonnumeric and may be any figurative constant except ALL.

The comparision operands of the TALLYING option (literal-1 or identifier-3, and so on) and/or REPLACING option (literal-3 or identifier-5, and so on) are compared in the left-to-right order specified in the INSPECT statement. A maximum of 15 comparison operands may be specified for each REPLACING and each TALLYING option.

When the TALLYING/REPLACING operands are the compared operands, the following comparison rules apply:

1. When both the TALLYING and REPLACING options are specified, the INSPECT statement is executed as if an INSPECT TALLYING statement were specified and immediately followed by an INSPECT REPLACING statement.

2. The first operand is compared with an equal number of leftmost contiguous characters in the inspected item. The operand matches the inspected characters only if both are equal, character-for-character.

3. If no match occurs for the first operand, the comparison is repeated for each successive operand until either a match is found or all operands have been acted upon.

4. If a match is found, tallying or replacing takes place as described in TALLYING/REPLACING option descriptions. In the inspected item, the first character following the rightmost matching character is now considered the leftmost character position. The process described in comparison rules 2 and 3 is then repeated.

5. If no match is found, the first character in the inspected item following the leftmost inspected character is now considered the leftmost character position. The process described in comparison rules 2 and 3 is then repeated.

6. The actions taken in comparison rules 1 through 5—which are defined as the comparison cycle—are repeated until the rightmost character in the inspected item has either been matched or has been considered as the leftmost character position. Inspection then terminates.

Figure 5-16 illustrates INSPECT statement comparisons.

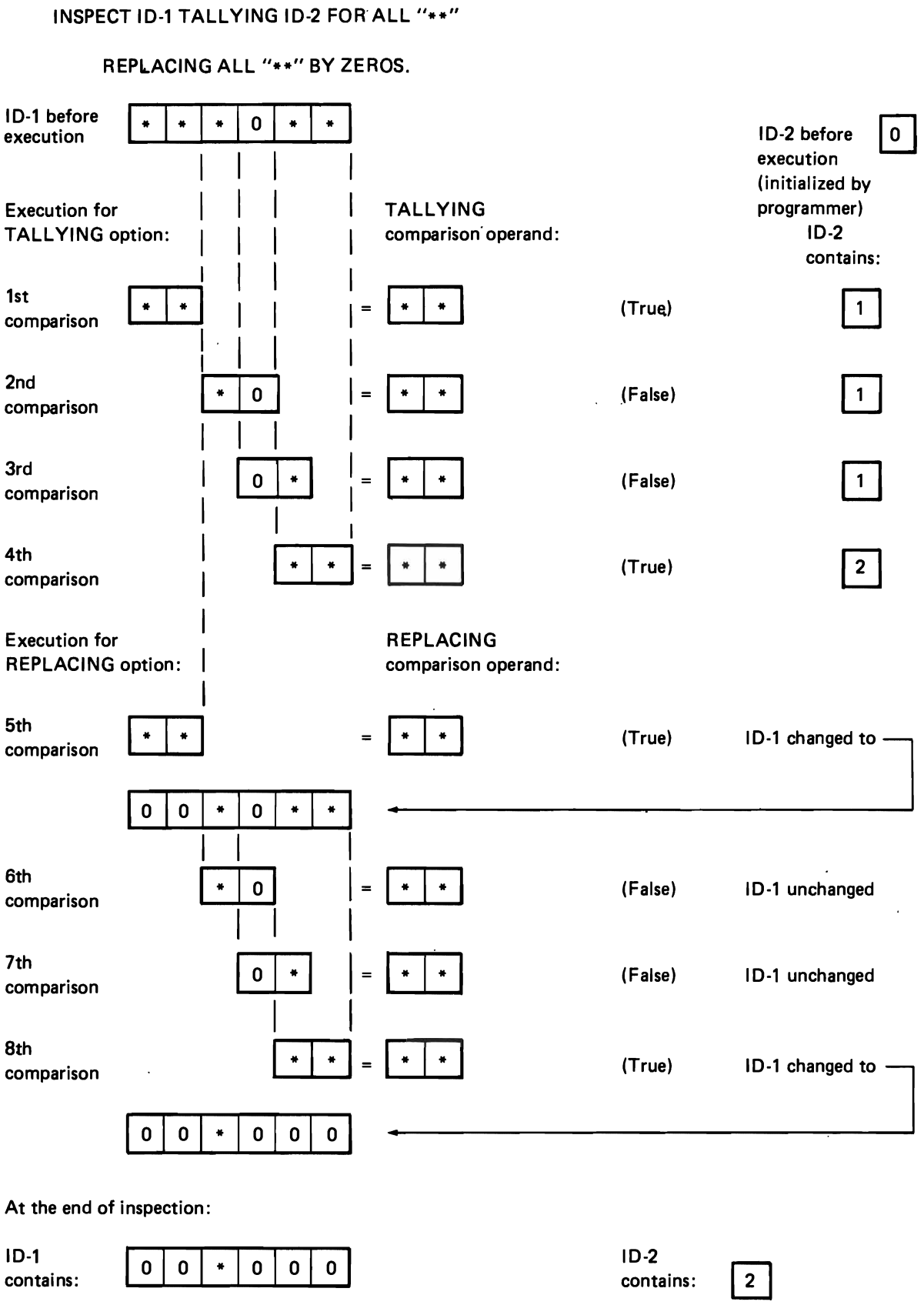


Figure 5-16. INSPECT Statement Execution Results

Note: When the BEFORE/AFTER option is specified, the preceding rules are modified as described in the BEFORE/AFTER option description.

INSPECT Statement Example

The following example shows an INSPECT statement.

```
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  ID-1 PIC X(10) VALUE 'ACADEMIANS'.
 01  CONTR-1 PIC 99 VALUE 00.
 01  CONTR-2 PIC 99 VALUE ZEROS.

 PROCEDURE DIVISION.
*    THIS ILLUSTRATES AN INSPECT STATEMENT WITH 2 VARIABLES.
 100-BEGIN-PROCESSING.
     DISPLAY CONTR-1 SPACE CONTR-2.
 101-MAINLINE-PROCESSING.
     PERFORM COUNT-IT THRU COUNT-EXIT.
     STOP RUN.
 COUNT-IT.
     INSPECT ID-1
         TALLYING CONTR-1 FOR CHARACTERS


     BEFORE INITIAL 'AD'  CONTR-2 FOR ALL 'MIANS'.
 DISPLAY-COUNTS.
     DISPLAY 'CONTR-1 = ' CONTR-1.
     DISPLAY 'CONTR-2 = ' CONTR-2.

     DISPLAY '**********EOJ**********'.
 COUNT-EXIT. EXIT.
```

Note: **The keywords BEFORE and AFTER should not be used in the same statement.**

RESULTANT OUTPUT

```
00 00
CONTR-1 = 02
CONTR-2 = 01
**********EOJ**********
```

TALLYING Option

Identifier-2 is the tallying field and must be an elementary integer item defined without the symbol P in its PICTURE character-string. It is the programmer's responsibility to initialize identifier-2 before the INSPECT statement is executed.

Identifier-3 or literal-1 is the comparison operand. If the comparison operand is a figurative constant, it is considered to be a one-character nonnumeric literal.

When the BEFORE/AFTER option is not specified, the following actions take place when the INSPECT TALLYING statement is executed:

- If the ALL phrase is specified, the tallying field is increased by one for each nonoverlapping occurrence in the inspected item of the comparison operand. This process begins at the leftmost character position and continues to the rightmost.
- If the LEADING phrase is specified, the tallying field is increased by one for each contiguous nonoverlapping occurrence of the comparison operand in the inspected item, provided the leftmost such occurrence is at the point where comparison began in the first comparison cycle for which the comparison operand is eligible to participate.
- If the CHARACTERS phrase is specified, the tallying field is increased by one for each character (including the space character) in the inspected item. Thus, execution of the INSPECT TALLYING statement increases the value in the tallying field by the number of characters in the inspected item.

REPLACING Option

Identifier-5 or literal-3 is the comparision operand. Identifier-6 or literal-4 is the replacement field.

The comparison operand and the replacement field must be the same length. The following replacement rules apply:

- If the comparison operand is a figurative constant, it is considered to be a one-character nonnumeric literal. Each character in the inspected item equivalent to the figurative constant is replaced by the single-character replacement field, which must be one character in length.
- If the replacement field is a figurative constant, it is considered to be the same length as the comparison operand . Each nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the replacement field.
- When the comparison operand and replacement fields are character-strings, each nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the character-string specified in the replacement field.
- Once replacement has occurred in a given character position in the inspected item, no further replacement for that character position is made in this execution of the INSPECT statement.

When the BEFORE/AFTER option is not specified, the following actions take place when the INSPECT REPLACING statement is executed:

- If the CHARACTERS phrase is specified, the replacement field must be 1 character in length. Each character in the inspected field is replaced by the replacement field. This process begins at the leftmost character and continues to the rightmost.
- If the ALL phrase is specified, each nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the replacement field, beginning at the leftmost character and continuing to the rightmost.

- If the LEADING phrase is specified, each contiguous nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the replacement field, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle for which this replacement field is eligible to participate.

- If the FIRST phrase is specified, the leftmost occurrence of the comparison operand in the inspected item is replaced by the replacement field.

BEFORE/AFTER Options

When either of these options is specified, the preceding rules for counting and replacing are modified.

Identifier-4, identifier-7, literal-2, and literal-5 are delimiters. Counting and/or replacement of the inspected item is bounded by their presence; however, the delimiters themselves are not counted or replaced.

In the TALLYING option, if the delimiter (literal-2) is a figurative constant it is considered to be 1 character in length.

In the REPLACING option, if the CHARACTERS phrase is specified, the delimiter (literal-5 or identifier-7) must be 1 character in length.

When the BEFORE option is specified, tallying and/or replacement of the inspected item begins at the leftmost character and continues until the first occurrence of the delimiter is encountered. If no delimiter is present in the inspected item, counting and/or replacement continues to the rightmost character.

When the AFTER option is specified, counting and/or replacement of the inspected item begins with the first character to the right of the delimiter and continues to the rightmost character in the inspected item. If no delimiter is present in the inspected item, no counting or replacement takes place.

INSPECT Statement Examples

The following examples illustrate some uses of the INSPECT statement. In all instances, the programmer has initialized the COUNTR field to zero before the INSPECT statement is executed.

INSPECT ID-1 REPLACING CHARACTERS BY ZERO.

ID-1 Before	COUNTR After	ID-1 After
1234567	0	0000000
HIJKLMN	0	0000000

INSPECT ID-1 TALLYING COUNTR FOR CHARACTERS REPLACING CHARACTERS BY SPACES.

ID-1 Before	COUNTR After	ID-1 After
1234567	7	
HIJKLMN	7	

INSPECT ID-1 REPLACING CHARACTERS BY ZEROS BEFORE INITIAL QUOTE.

ID-1 Before	COUNTR After	ID-1 After
456'ABEL	0	000'ABEL
ANDES'12	0	00000'12
'TWAS BR	0	'TWAS BR

INSPECT ID-1 TALLYING COUNTR FOR CHARACTERS AFTER INITIAL 'S'REPLACING ALL 'A' BY 'O'.

ID-1 Before	COUNTR After	ID-1 After
ANSELM	3	ONSELM
SACKET	5	SOCKET
PASSED	3	POSSED

INSPECT ID-1 TALLYING COUNTR FOR LEADING '0' REPLACING FIRST 'A' BY '2' AFTER INITIAL 'C'.

ID-1 Before	COUNTR After	ID-1 After
00ACADEMY00	2	00AC2DEMY00
0000ALABAMA	4	0000ALABAMA
CHATHAM0000	0	CH2THAM0000

Programming Notes

The INSPECT statement is useful for filling portions or all of a data item with spaces or zeros. It is also useful for counting the number of times a specific character (for example, zero, space, asterisk) occurs in a data item. In addition, it can be used to translate characters from one collating sequence to another.

MOVE Statement

The MOVE statement transfers data from one area of storage to one or more other areas. The formats of the MOVE statement are as follows:

Format 1

```
MOVE { identifier-1 } TO identifier-2 [, identifier-3] . . .
     { literal      }
```

Format 2

```
MOVE { CORRESPONDING } identifier-1 TO identifier-2
     { CORR          }
```

Identifier-1 and literal-1 are the sending areas. Identifier-2, identifier-3, and so on are the receiving areas.

When Format 1 is specified, the identifiers may be either group or elementary items. The data in the sending area is moved into the first receiving area (identifier-2); then it is moved into the second receiving area (identifier-3), and so on.

When Format 2 is specified, both identifiers must be group items. CORR is an abbreviation for, and equivalent to, CORRESPONDING. When CORRESPONDING is specified, selected items in identifier-1 are moved to identifier-2 according to the rules for the CORRESPONDING option. The results are the same as if each pair of CORRESPONDING identifiers had been referred to in a separate MOVE statement.

An index data item cannot be specified in a MOVE statement. Any subscripting or indexing associated with the sending item is evaluated only once: immediately before the data is moved to the first receiving field. Any subscripting or indexing associated with the receiving items is evaluated immediately before the data is moved into the receiving field.

For example, the result of the statement:

```
MOVE A (B) TO B, C (B).
```

is equivalent to

```
MOVE A (B) TO TEMP.
MOVE TEMP TO B.
MOVE TEMP TO C (B).
```

where TEMP has been defined as an intermediate result item. The subscript B changed in value between the time the first move took place, and the final move to C (B) is executed.

After execution of a MOVE statement, the sending field(s) contains the same data as before execution.

Elementary Moves

An elementary move is one in which both the sending and receiving items are elementary items. Each elementary item belongs to one of the following categories:

- Numeric–includes numeric data items, numeric literals, and the figurative constant ZERO/ZEROS/ZEROES when the receiving item is numeric.
- Alphabetic–includes alphabetic data items and the figurative constant SPACE/SPACES.
- Alphanumeric–includes alphanumeric data items, nonnumeric literals, and all figurative constants except ZERO and SPACE.
- Alphanumeric edited–includes alphanumeric edited data items.
- Numeric edited–includes numeric edited data items.
- Boolean–includes Boolean data items, Boolean literals, and the figurative constant ZERO/ZEROS/ZEROES when the receiving item is Boolean.

Valid elementary moves are executed according to the following rules:

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

- For an alphanumeric, alphanumeric edited, or alphabetic receiving item:
  - Justification and any necessary space filling take place as described in the JUSTIFIED clause. Unused character positions are filled with spaces.
  - If the size of the sending item is greater than the size of the receiving item, excess characters at the right are truncated after the receiving item is filled.
  - If the sending item has an operational sign, the absolute value is used. If the operational sign occupies a separate character, that character is not moved, and the size of the sending item is considered to be one less than its actual size.
  - If the sending item is Boolean, and the receiving item is alphanumeric or alphanumeric edited, no data conversion takes place.

- For a numeric or numeric edited receiving item:
  - Alignment by decimal point and any necessary zero filling take place as described under *Standard Alignment Rules* in Chapter 4, except where zeros are replaced because of editing requirements.
  - The absolute value of the sending item is used if the receiving item has no operational sign.
  - If the sending item has more digits to the left or right of the decimal point than the receiving item can contain, excess digits are truncated.
  - The results at object time may be unpredictable if the sending item contains any nonnumeric characters.

- For a Boolean receiving item:
  - There is no data conversion.
  - The source field must be either alphanumeric or Boolean.
  - Execution of the MOVE statement does not affect the association of an indicator number to the data name.

Note: If the receiving field is alphanumeric or numeric edited, and the sending field is a scaled integer (that is, it has a P as the rightmost character in its PICTURE character-string), the scaling positions are treated as trailing zeros when the MOVE statement is executed.

Figure 5-17 shows valid and invalid elementary moves for each category.

Sending Item Category	Receiving Item Category						
	Alphabetic	Alphanumeric	Alphanumeric Edited	Numeric Integer	Numeric Noninteger	Numeric Edited	Boolean
Alphabetic and SPACE	YES	YES	YES	NO	NO	NO	NO
Alphanumeric and Figurative constant[1]	YES	YES	YES	YES	YES	YES	YES
Alphanumeric Edited	YES	YES	YES	NO	NO	NO	NO
Numeric Integer[2] and ZERO	NO	YES	YES	YES	YES	YES	NO
Numeric Noninteger[2]	NO	NO	NO	YES	YES	YES	NO
Numeric Edited	NO	YES	YES	NO	NO	NO	NO
Boolean[3]	NO	YES	YES	NO	NO	NO	YES

YES = move is valid
NO = move is invalid

[1] Includes nonnumeric literals and all figurative constants but SPACE and ZERO.
[2] Includes numeric literals
[3] Includes the figurative constants ZERO and ALL Boolean-literal.

Figure 5-17. Valid and Invalid Elementary Moves

Group Moves

A group move is one in which one or both of the sending and receiving fields are a group item. A group move is treated exactly as though it were an alphanumeric elementary move except that data is not converted from one form of internal representation to another. In a group move, the receiving area is filled without consideration for the individual elementary items contained within either the sending area or the receiving area.

STRING Statement

The STRING statement gives the programmer the ability to concatenate the partial or complete contents of two or more data items in a single data item.

Format

```
STRING { identifier-1 } [ ,identifier-2 ] ... DELIMITED BY { identifier-3 }
       { literal-1    }   [ ,literal-2    ]               { literal-3    }
                                                          { SIZE         }

    [ , { identifier-4 } [ ,identifier-5 ] ... DELIMITED BY { identifier-6 } ] ...
        { literal-4    }   [ ,literal-5    ]               { literal-6    }
                                                           { SIZE         }

  INTO identifier-7 [ WITH POINTER identifier-8 ]

  [ ON OVERFLOW imperative-statement ]
```

Each literal must be a nonnumeric literal; each may be any figurative constant without the optional word ALL. When a figurative constant is specified, it is considered a 1-character nonnumeric literal.

All identifiers except identifier-8 (the POINTER item) must have USAGE DISPLAY, explicitly or implicitly.

The sending fields are identifier-1, identifier-2, identifier-4, identifier-5, or their corresponding literals.

The receiving field is identifier-7, which must be an elementary alphanumeric item without editing symbols and without the JUSTIFIED clause in its description.

The delimiters are identifier-3, identifier-6, or their corresponding literals, or the key word SIZE. The delimiters specify the character(s) delimiting the data to be transferred; when SIZE is specified, the complete sending area is transferred.

When the sending field or any of the delimiters are elementary numeric items, they must be described as integers, and their PICTURE character-strings must not contain the symbol P.

The pointer field is identifier-8, which must be an elementary integer data item large enough to contain a value equal to the length of the receiving area plus one. The pointer field must not contain the symbol P in its PICTURE character-string.

STRING Statement Execution

When the STRING statement is executed, data is transferred from the sending fields to the receiving field. The order in which sending fields are processed is the order in which they are specified. The following rules apply:

- Characters from the sending fields are transferred to the receiving field according to the rules for alphanumeric to alphanumeric elementary moves except that no space filling is provided.
- When the DELIMITED BY identifier/literal is specified, the contents of each sending item are transferred character by character beginning with the leftmost and continuing until either a delimiter for this sending field is reached (the delimiter itself is not transferred) or the rightmost character of this sending field has been transferred.
- When DELIMITED BY SIZE is specified, each sending field is transferred in its entirety to the receiving field.
- When the receiving field is filled or when all the sending fields have been processed, the operation is ended.

- When the POINTER option is specified, an explicit pointer field is available to the COBOL user to control placement of data in the receiving field. The user must set the explicit pointer's initial value, which must not be less than one and not more than the character count of the receiving field. The pointer field must be defined as large enough to contain a value equal to the length of the receiving field plus 1; this precludes arithmetic overflow when the system updates the pointer at the end of the transfer.

- When the POINTER option is not specified, no pointer is available to the user. However, an implicit pointer with an initial value of one is used by the system.

- When the STRING statement is executed, the initial pointer value (explicit or implicit) points to the first character position within the receiving field into which data is to be transferred. Beginning at that position, data is then positioned character by character from left to right. After each character is positioned, the explicit or implicit pointer is incremented by one. The value in the pointer field is changed only in this manner. At the end of processing, the pointer value always indicates one character beyond the last character transferred into the receiving field.

- If, at any time during or after initiation of STRING statement execution, the pointer value (explicit or implicit) is less than one or exceeds a value equal to the length of the receiving field, no more data is transferred into the receiving field and, if specified, the ON OVERFLOW imperative-statement is executed. (The ON OVERFLOW statement is not executed unless there was an attempt to move in one or more characters beyond the end of identifier-7.)

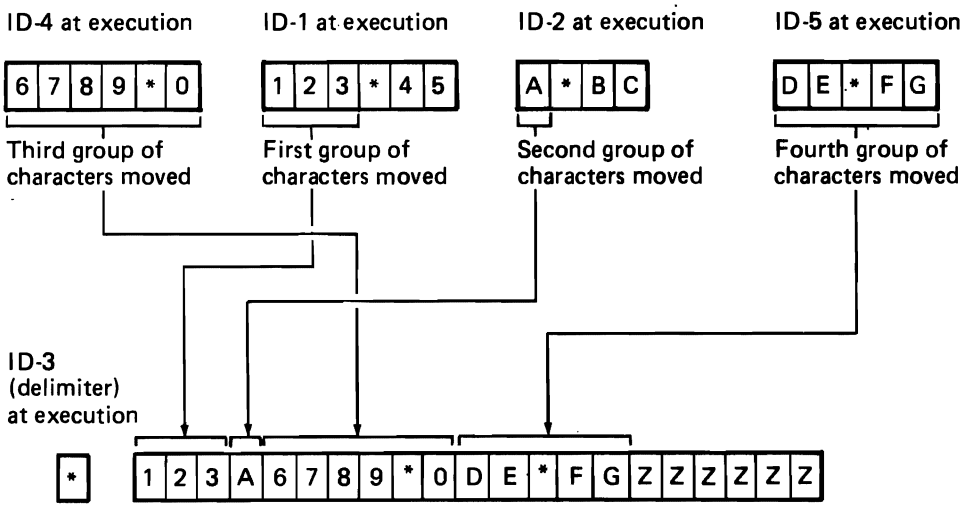
- If the ON OVERFLOW option is not specified, then when the preceding conditions occur, control passes to the next executable statement.

After STRING statement execution is completed, only that part of the receiving field into which data was transferred is changed. The rest of the receiving field contains the data that was present before this execution of the STRING statement. Figure 5-18 illustrates the rules of execution for the STRING statement.

STRING statement to be executed:

```
STRING  ID-1   ID-2 DELIMITED BY ID-3
        ID-4   ID-5 DELIMITED BY SIZE
  INTO  ID-7   WITH POINTER ID-8.
```

Results:



ID-8
(pointer)
after execution

1	6

(initialized to 01 before execution)

Figure 5-18. STRING Statement Execution Results

STRING Statement Example

The following example illustrates some of the considerations that apply to the STRING statement.

In the Data Division, the programmer has defined the following fields:

```
01  RPT-LINE      PICTURE X(120).
01  LINE-POS      PICTURE 99.
01  LINE-NO       PICTURE 9(5) VALUE 1.
01  DEC-POINT     PICTURE X VALUE '.'.
```

In the File Section, he has defined the following input record:

```
01  RCD-01.
    05  CUST-INFO.
        10  CUST-NAME     PICTURE X(15).
        10  CUST-ADDR     PICTURE X(34).
    05  BILL-INFO.
        10  INV-NO        PICTURE X(6).
        10  INV-AMT       PICTURE $$,$$$.99.
        10  AMT-PAID      PICTURE $$,$$$.99.
        10  DATE-PAID     PICTURE X(8).
        10  BAL-DUE       PICTURE $$,$$$.99.
        10  DATE-DUE      PICTURE X(8).
```

The programmer wants to construct an output line consisting of portions of the information from RCD-01. The line is to consist of a line number, customer name and address, invoice number, date due, and balance due, truncated to the dollar figure shown.

The record as read in contains the following information:

```
J.B.␣SMITH␣␣␣␣␣
444␣SPRING␣ST.,␣CHICAGO,␣ILL.␣␣␣␣␣
A14275
$4,736.85
$2,400.00
09/22/76
$2,336.85
10/22/76
```

In the Procedure Division, the programmer initializes RPT-LINE to SPACES and sets LINE-POS (which is to be used as the POINTER field) to 4. Then he issues this STRING statement:

```
STRING LINE-NO SPACE CUST-INFO
    SPACE INV-NO SPACE DATE-DUE
    SPACE DELIMITED BY SIZE
    BAL-DUE DELIMITED BY DEC-POINT
    INTO RPT-LINE WITH POINTER LINE-POS.
```

When the statement is executed, the following actions take place:

1. The field LINE-NO is moved into positions 4 through 8 of RPT-LINE.
2. A space is moved into position 9.
3. The group item CUST-INFO is moved into positions 10 through 58.
4. A space is moved into position 59.
5. INV-NO is moved into positions 60 through 65.
6. A space is moved into position 66.
7. DATE-DUE is moved into positions 67 through 74.
8. A space is moved into position 75.
9. The portion of BAL-DUE that precedes the decimal point is moved into positions 76 through 81.

After the STRING statement has been executed, RPT-LINE appears as shown in Figure 5-19.

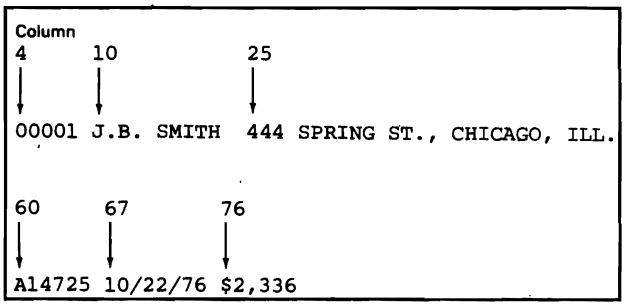


Figure 5-19. STRING Statement Example Output Data

Programming Notes

One STRING statement can be written instead of a series of MOVE statements.

UNSTRING Statement

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

Format

```
UNSTRING identifier-1
  [ DELIMITED BY [ ALL ] { identifier-2 } [ , OR [ ALL ] { identifier-3 } ] ... ]
                         { literal-1    }               { literal-2    }
    INTO identifier-4 [ , DELIMITER IN identifier-5 ] [ , COUNT IN identifier-6 ]
      [ , identifier-7 [ , DELIMITER IN identifier-8 ] [ , COUNT IN identifier-9 ] ] ...
  [ WITH POINTER identifier-10 ] [ TALLYING IN identifier-11 ]
  [ ON OVERFLOW imperative-statement ]
```

Each literal must be a nonnumeric literal; each may be any figurative constant except ALL literal. When a figurative constant is specified, it is considered to be a 1-character nonnumeric literal.

Sending Field

Identifier-1 is the sending field. It must be an alphanumeric data item. Data is transferred from this field to the receiving fields.

DELIMITED BY Option: This option specifies delimiters within identifier-1 that control the data transfer.

The delimiters are identifier-2, identifier-3, or their corresponding literals. Each identifier or literal specified represents one delimiter. No more than 15 delimiters may be specified. Each must be an alphanumeric data item.

If a delimiter contains two or more characters, it is recognized in the sending field only if the delimiter characters are contiguous and, in the sequence specified, in the delimiter item.

When two or more delimiters are specified, an OR condition exists and each nonoverlapping occurrence of any one of the delimiters is recognized in the sending field in the sequence specified. For example, if DELIMITED BY AB OR BC is specified, then an occurrence of either AB or BC in the sending field is considered a delimiter. An occurrence of ABC is considered an occurrence of AB, and the search for another delimiter resumes with C.

When the DELIMITED BY ALL option is not specified, and two or more contiguous occurrences of any delimiter are encountered, the current data receiving field is filled with spaces or zeros according to the description of the data receiving field.

When the DELIMITED BY ALL option is specified, one or more contiguous occurrences of any delimiter are treated as if they were only one occurrence, and this one occurrence is moved to the delimiter receiving field (if specified). The delimiting characters in the sending field are treated as an elementary alphanumeric item and are moved into the current delimiter receiving field according to the rules of the MOVE statement.

The DELIMITER IN and COUNT IN options can be specified only if the DELIMITED BY option is specified.

Data Receiving Fields

Identifier-4, identifier-7, and so on, are the data receiving fields and must have USAGE DISPLAY. These fields can be defined as:

- Alphabetic (without the symbol B in the PICTURE string)
- Alphanumeric
- Numeric (without the symbol P in the PICTURE string)

These fields must not be defined as alphanumeric edited or numeric edited items. Data is transferred to these fields from the sending field.

DELIMITER IN Option: The delimiter receiving fields are identifier-5, identifier-8, and so on. These identifiers must be alphanumeric.

COUNT IN Option: The data-count fields for each data transfer are identifier-6, identifier-9, and so on. Each field holds the count of delimited characters in the sending field to be transferred to this receiving field; the delimiters are not included in this count.

POINTER Option: The pointer field is identifier-10; it contains a value that indicates the relative starting position in the sending field. When this option is specified, the user must initialize this field before execution of the UNSTRING statement to a value that is not less than one and not greater than the count of the sending field.

TALLYING Option: The field-count is identifier-11; it is incremented by the number of data receiving fields acted upon in this execution of the UNSTRING statement. When this option is specified, the user must initialize this field before execution of the UNSTRING statement.

The data-count fields, the pointer field, and the field-count field must each be integer items without the symbol P in the PICTURE character-strings.

UNSTRING Statement Execution

When the UNSTRING statement is initiated, the current data receiving field is identifier-4. Data is transferred from the sending field to the current data receiving field according to the following rules:

- If the POINTER option is not specified, the sending field character-string is examined beginning with the leftmost character. If the POINTER option is specified, the field is examined beginning at the relative character position specified by the value in the pointer field.
- If the DELIMITED BY option is specified, the examination proceeds left to right character by character until a delimiter is encountered. If the end of the sending field is reached before a delimiter is found, the examination ends with the last character in the sending field.
- If the DELIMITED BY option is not specified, the number of characters examined is equal to the size of the current data receiving field, which depends on its data category:
  - If the receiving field is alphanumeric or alphabetic, the number of characters examined is equal to the number of characters in the current receiving field.
  - If the receiving field is numeric, the number of characters examined is equal to the number of characters in the integer portion of the current receiving field.
  - If the receiving field is described with the SIGN IS SEPARATE clause, the characters examined are one fewer than the size of the current receiving field.
  - If the receiving field is described as a variable-length data item, the number of characters examined is determined by the current size of the current receiving field.
- The examined characters (excluding any delimiter characters) are treated as an alphanumeric elementary item, and are moved into the current data receiving field according to the rules for the MOVE statement.

- If the DELIMITER IN option is specified, the delimiting characters in the sending field are treated as an elementary alphanumeric item and are moved to the current delimiter receiving field according to the rules for the MOVE statement. If the delimiting condition is the end of the sending field, the current delimiter receiving field is filled with spaces.

- If the COUNT IN option is specified, a value equal to the number of examined characters (excluding any delimiters) is moved into the data count field, according to the rules for an elementary move.

- If the DELIMITED BY option is specified, the sending field is further examined, beginning with the first character to the right of the delimiter.

- If the DELIMITED BY option is not specified, the sending field is further examined, beginning with the first character to the right of the last character examined.

- After data is transferred to the first data receiving field (identifier-4), the current data receiving field becomes identifier-7. For each succeeding current data receiving field, the preceding procedure is repeated – either until all of the characters in the sending field have been transferred, or until there are no more unfilled data receiving fields.

- When the POINTER option is specified, the contents of the pointer field behaves as if incremented by one for each examined character in the sending field. When this execution of the UNSTRING statement is completed, the pointer field contains a value equal to its initial value plus the number of characters examined in the sending field.

- When the TALLYING option is specified and the execution of the UNSTRING statement is completed, the tallying identifier contains a value equal to the initial value plus the number of data receiving areas acted upon; this count includes any null fields.

- When an overflow condition exists, the execution of the UNSTRING statement is terminated. If the ON OVERFLOW option has been specified, that imperative-statement is executed. If the ON OVERFLOW option has not been specified, control passes to the next executable statement. An overflow condition exists when:
  - An UNSTRING statement is initiated and the value in the pointer field is less than 1 or greater than the length of the sending field.
  - Or, all data receiving fields have been acted upon during UNSTRING statement execution, and the sending field still contains unexamined characters.

If any of the UNSTRING statement identifiers are subscripted or indexed, the subscripts and indexes are evaluated as follows:

- Any subscripting or indexing associated with the sending field, the pointer field, or the field-count field is evaluated only once – immediately before any data is transferred.

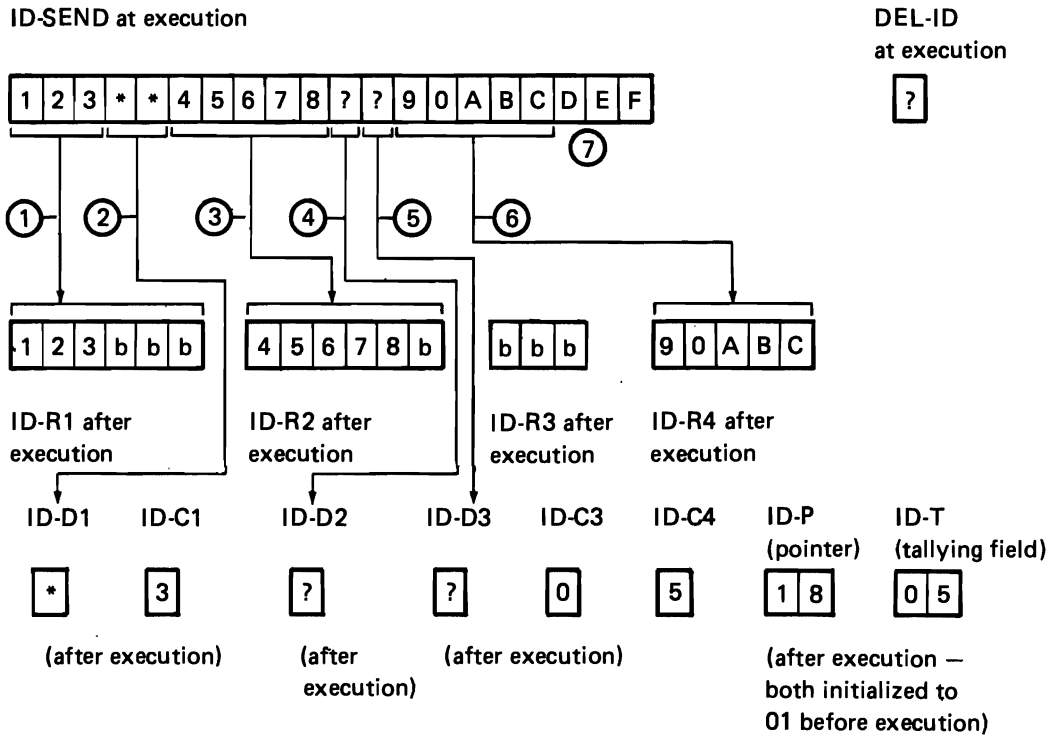
- Any subscripting or indexing associated with the delimiters, the data and delimiter receiving fields or the data-count fields, is evaluated immediately before the transfer of data into the affected data item.

Figure 5-20 illustrates the rules of execution for the UNSTRING statement.

The following UNSTRING statement has the execution results shown:

```
UNSTRING  ID-SEND DELIMITED BY DEL-ID OR ALL '*'
      INTO  ID-R1 DELIMITER IN ID-D1 COUNT IN ID-C1
            ID-R2 DELIMITER IN ID-D2
            ID-R3 DELIMITER IN ID-D3 COUNT IN ID-C3
            ID-R4 COUNT IN ID-C4
      WITH POINTER ID-P
      TALLYING IN ID-T
      ON OVERFLOW GO TO OFLOW-EXIT.
```

(All the data receiving fields are defined as alphanumeric)



The order of execution is:

(1) Three characters are placed in ID-R1.

(2) Because ALL * is specified, one * is placed in ID-D1.

(3) Five characters are placed in ID-R2.

(4) A ? is placed in ID-D2. The current receiving field is now ID-R3.

(5) A ? is placed in ID-D3; ID-R3 is filled with spaces; no characters are transferred, so 0 is placed in ID-C3.

(6) No delimiter is encountered before 5 characters fill ID-R4; 5 is placed in ID-C4.

(7) ID-P is updated to 18; ID-T is updated to 05. There are still untransferred characters still existing in ID-SEND, and so the ON OVERFLOW exit is taken.

Figure 5-20. UNSTRING Statement Execution Results

UNSTRING Statement Example

The following example illustrates some of the considerations that apply to the UNSTRING statement.

In the Data Division, the programmer has defined the following input record to be acted upon by the UNSTRING statement:

```
01  INV-RCD.
    05  CONTROL-CHARS      PIC XX.
    05  ITEM-INDENT        PIC X(20).
    05  FILLER             PIC X.
    05  INV-CODE           PIC X(10).
    05  FILLER             PIC X.
    05  NO-UNITS           PIC 9(6).
    05  FILLER             PIC X.
    05  PRICE-PER-M        PIC 99999.
    05  FILLER             PIC X.
    05  RTL-AMT            PIC 9(6).99.
```

The next two records are defined as receiving fields for the UNSTRING statement. DISPLAY-REC is to be used for printed output. WORK-REC is to be used for further internal processing.

```
01  DISPLAY-REC.
    05  INV-NO             PIC X(6).
    05  FILLER             PIC X VALUE SPACE.
    05  ITEM-NAME          PIC X(20).
    05  FILLER             PIC X VALUE SPACE.
    05  DISPLAY-DOLS       PIC 9(6).

01  WORK-REC.
    05  M-UNITS            PIC 9(6).
    05  FIELD-A            PIC 9(6).
    05  WK-PRICE
        REDEFINES
        FIELD-A             PIC 9999V99.
    05  INV-CLASS          PIC X(3).
```

The programmer has also defined the following fields for use as control fields in the UNSTRING statement.

```
77  DBY-1                  PIC X, VALUE IS '.'.
77  CTR-1                  PIC 99, VALUE IS ZERO.
77  CTR-2                  PIC 99, VALUE IS ZERO.
77  CTR-3                  PIC 99, VALUE IS ZERO.
77  CTR-4                  PIC 99, VALUE IS ZERO.
77  DLTR-1                 PIC X.
77  DLTR-2                 PIC X.
77  CHAR-CT                PIC 99, VALUE IS 3.
77  FLDS-FILLED            PIC 99, VALUE IS ZERO.
```

In the Procedure Division, the programmer writes the following UNSTRING statement to move subfields of INV-RCD to the subfields of DISPLAY-REC and WORK-REC:

```
UNSTRING INV-RCD DELIMITED BY
    ALL SPACES OR '/' OR DBY-1
    INTO ITEM-NAME COUNT IN CTR-1
    INV-NO DELIMITER IN DLTR-1 COUNT IN CTR-2
    INV-CLASS
    M-UNITS COUNT IN CTR-3
    FIELD-A
    DISPLAY-DOLS DELIMITER IN
    DLTR-2 COUNT IN CTR-4
    WITH POINTER CHAR-CT
    TALLYING IN FLDS-FILLED
    ON OVERFLOW GO TO UNSTRING-COMPLETE.
```

Before the UNSTRING statement is issued, the programmer places the value 3 in the CHAR-CT (the POINTER item), so as not to work with the two control characters at the beginning of INV-RCD. In DBY-1, a period is placed for use as a delimiter, and in FLDS-FILLED (the TALLYING item) the value 0 is placed. The following data is then read into INV-RCD as shown in Figure 5-21.

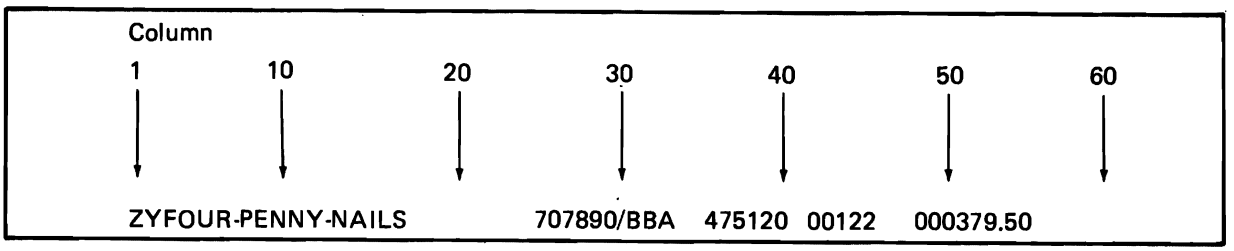


Figure 5-21. UNSTRING Statement Example – Input Data

When the UNSTRING statement is executed, the following actions take place:

1. Positions 3 through 18 (FOUR-PENNY-NAILS) of INV-RCD are placed in ITEM-NAME, left-justified within the area, and the unused character positions are padded with spaces. The value 16 is placed in CTR-1.

2. Because ALL SPACES is specified as a delimiter, the five contiguous SPACE characters are considered to be one occurrence of the delimiter.

3. Positions 24 through 29 (707890) are placed in INV-NO. The delimiter character / is placed in DLTR-1, and the value 6 is placed in CTR-2.

4. Positions 31 through 33 are placed in INV-CLASS. The delimiter is a SPACE, but because no field has been defined as a receiving area for delimiters, the SPACE is merely bypassed.

5. Positions 35 through 40 (475120) are examined and are placed in M-UNITS. The delimiter is a SPACE, but because no receiving field has been defined as a receiving area for delimiters, the SPACE is bypassed. The value 6 is placed in CTR-3.

6. Positions 42 through 46 (00122) are placed in FIELD-A and right-justified within the area. The high-order digit position is filled with a 0 (zero). The delimiter is a SPACE, but because no field has been defined as a receiving area for delimiters, the SPACE is bypassed.

7. Positions 48 through 53 (000379) are placed in DISPLAY-DOLS. The period (.) delimiter character is placed in DLTR-2, and the value 6 is placed in CTR-4.

8. Because all receiving fields have been acted upon and two characters of data in INV-RCD have not been examined, the ON OVERFLOW exit is taken, and execution of the UNSTRING statement is completed.

At the end of execution of the UNSTRING statement, DISPLAY-REC contains the following data:

```
707890 FOUR-PENNY-NAILS 000379
```

WORK-REC contains the following data:

```
475120000122BBA
```

CHAR-CT (the POINTER field) contains the value 55, and FLD-FILLED (the TALLYING field) contains the value 6.

Programming Notes

One UNSTRING statement can be written instead of a series of MOVE statements.

PROCEDURE BRANCHING STATEMENTS

Statements, sentences, and paragraphs in the Procedure Division are ordinarily executed sequentially. The procedure branching statements allow alterations in the sequence. The procedure branching statements are: ALTER, EXIT, GO TO, PERFORM, and STOP.

ALTER Statement

The ALTER statement changes the transfer point specified in a GO TO statement.

Format

```
ALTER procedure-name-1 TO [PROCEED TO] procedure-name-2
    [, procedure-name-3 TO [PROCEED TO] procedure-name-4 ] . . .
```

Procedure-name-1, procedure-name-3, and so on, must each name a Procedure Division paragraph that contains only one sentence. That sentence must be a GO TO statement without the DEPENDING ON option.

Procedure-name-2, procedure-name-4, and so on, must each name a Procedure Division section or paragraph.

ALTER statement execution modifies the GO TO statement in the paragraph named by procedure-name-1, procedure-name-3, and so on. Subsequent executions of the modified GO TO statement(s) cause control to be transferred to procedure-name-2, and (if specified) procedure-name-4, and so on. For example:

```
PARAGRAPH-1.
   GO TO BYPASS-PARAGRAPH.
PARAGRAPH-1A.
   .
   .
BYPASS-PARAGRAPH.
   .
   .
   ALTER PARAGRAPH-1 TO PROCEED TO
         PARAGRAPH-2.
   .
   .
PARAGRAPH-2.
   .
   .
```

Before the ALTER statement is executed, when control reaches PARAGRAPH-1, the GO TO statement transfers control to BYPASS-PARAGRAPH. After execution of the ALTER statement, however, the next time control reaches PARAGRAPH-1, the GO TO statement transfers control to PARAGRAPH-2.

Programming Notes

The ALTER statement acts as a program switch, allowing, for example, one sequence of execution during initialization and another sequence during the bulk of file processing. Because altered GO TO statements are difficult to debug, it is preferable to test a switch, and based on the value of the switch, execute a particular code sequence.

Segmentation Information

A GO TO statement in a section whose priority is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different priority. All other uses of the ALTER statement are valid and are performed.

Modified GO TO statements in independent segments may sometimes be returned to their initial states. For further discussion, see *Segmentation-Procedure Division* in Chapter 6.

EXIT Statement

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

Format

```
EXIT [PROGRAM] .
```

The EXIT statement must appear in a sentence by itself, and this sentence must be the only sentence in the paragraph. The EXIT statement enables the user to assign a procedure-name to a given point in a program.

The EXIT statement has no other effect on the compilation or execution of the program.

The EXIT PROGRAM statement is discussed under *Subprogram Linkage Statements* in Chapter 6.

Programming Notes

The EXIT statement is useful for documenting the end point in a series of procedures. If an exit paragraph is written as the last paragraph in a Declarative procedure or a series of performed procedures, it identifies the point at which control will be transferred. When control reaches such an exit paragraph and the associated Declarative or PERFORM statement is active, control is transferred to the appropriate part of the Procedure Division. When control reaches such an exit paragraph and no associated PERFORM statement or Declarative procedure is active, control passes through the EXIT statement to the first statement of the next paragraph.

If an EXIT statement is not written, the end of the sequence is difficult to determine unless the user knows the logic of the program.

GO TO Statement

The GO TO statement transfers control from one part of the Procedure Division to another. The formats of the GO TO statement are as follows:

Format 1

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

Format 2

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

     DEPENDING ON identifier
```

Each procedure-name specified must name a paragraph or section in the Procedure Division. Identifier must name an elementary integer item.

Format 1—Unconditional GO TO

The GO TO statement causes control to be transferred to the first statement in the paragraph or section named in procedure-name-1 unless the GO TO statement has been modified by an ALTER statement.

When a Format 1 GO TO statement appears in a sequence of imperative statements, it must be the last statement in the sequence.

When a paragraph is referred to by an ALTER statement, the paragraph may consist only of a paragraph-name followed by a Format 1 GO TO statement.

If procedure-name-1 is not specified in a Format 1 GO TO statement, an ALTER statement must have been executed before the execution of the GO TO statement. The GO TO statement must immediately follow a paragraph-name and must be the only statement in the paragraph.

Format 2—Conditional GO TO

Control is transferred to one of a series of procedures, depending on the value of identifier. When identifier has a value of one, control is transferred to the first statement in the procedure named by procedure-name-1; if it has a value of two, control is transferred to the first statement in the procedure named by procedure-name-2, and so on.

If the value of identifier is anything other than a value within the range 1 through n (where n is the number of procedure-names specified in this GO TO statement), the GO TO statement is ignored. Instead, control passes to the next statement in the normal sequence of execution.

The maximum number of procedure-names permitted for a Format 2 GO TO statement is 99. The identifier field can be defined as containing up to 4 bytes.

PERFORM Statement

The PERFORM statement transfers control explicitly to one or more procedures and implicitly returns control to the next executable statement after execution of the specified procedure(s) is completed. The formats of the PERFORM statement are as follows:

Format 1

```
PERFORM procedure-name-1 [ { THROUGH } procedure-name-2 ]
                         { THRU    }
```

Format 2

```
PERFORM procedure-name-1 [ { THROUGH } procedure-name-2 ] { identifier-1 } TIMES
                         { THRU    }                     { integer-1    }
```

Format 3

```
PERFORM procedure-name-1 [ { THROUGH } procedure-name-2 ] UNTIL condition-1
                         { THRU    }
```

Format 4

```
PERFORM procedure-name-1 [ { THROUGH } procedure-name-2 ]
                         { THRU    }

    VARYING { identifier-1 } FROM { identifier-2 }
            { index-name-1 }      { index-name-2 }
                                  { literal-2    }

    BY { identifier-3 } UNTIL condition-1
       { literal-3    }

  [ AFTER { identifier-4 } FROM { identifier-5 }
          { index-name-4 }      { index-name-5 }
                                { literal-5    }

    BY { identifier-6 } UNTIL condition-2
       { literal-6    }

    [ AFTER { identifier-7 } FROM { identifier-8 }
            { index-name-7 }      { index-name-8 }
                                  { literal-8    }

    BY { identifier-9 } UNTIL condition-3 ] ]
       { literal-9    }
```

Each procedure-name must name a section or paragraph in the Procedure Division.

When both procedure-name-1 and procedure-name-2 are specified, if either is a procedure-name in a Declarative procedure, then both must be procedure-names in the same Declarative procedure.

Each identifier must name a numeric elementary item.

Each literal must be a numeric literal.

Whenever a PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1. Control is always returned to the statement following the PERFORM statement. The point from which this control is returned is determined as follows:

- If procedure-name-1 is a paragraph name and procedure-name-2 is not specified, the return is made after the execution of the last statement of procedure-name-1.

- If procedure-name-1 is a section name and procedure-name-2 is not specified, the return is made after the execution of the last sentence of the last paragraph in that section.

- If procedure-name-2 is specified and it is a paragraph name, the return is made after the execution of the last statement of that paragraph.

- If procedure-name-2 is specified and it is a section name, the return is made after the execution of the last sentence of the last paragraph in the section.

The only necessary relationship between procedure-name-1 and procedure-name-2 is that a consecutive sequence of operations is executed beginning at the procedure named by procedure-name-1 and ending with the execution of the procedure named by procedure-name-2.

When both procedure-name-1 and procedure-name-2 are specified, GO TO and PERFORM statements may appear within the sequence of statements contained in these paragraphs or sections. When only procedure-name-1 is specified, PERFORM statements may appear within the procedure. A GO TO statement may also appear, but should not refer to a procedure-name outside the range of procedure-name-1. If this is done, results are unpredictable and are not diagnosed.

When the performed procedures include another PERFORM statement, the sequence of procedures associated with the embedded PERFORM statement must be totally included in or totally excluded from the performed procedures of the first PERFORM statement. That is, an active PERFORM statement whose execution point begins within the range of performed procedures of another active PERFORM statement must not allow control to pass through the exit point of the other active PERFORM statement. In addition, two or more such active PERFORM statements must not have a common exit.

When control passes to the sequence of procedures by means other than a PERFORM statement, control passes through the exit point to the next executable statement as if no PERFORM statement referred to these procedures.

Figure 5-22 illustrates valid sequences of execution for PERFORM statements.

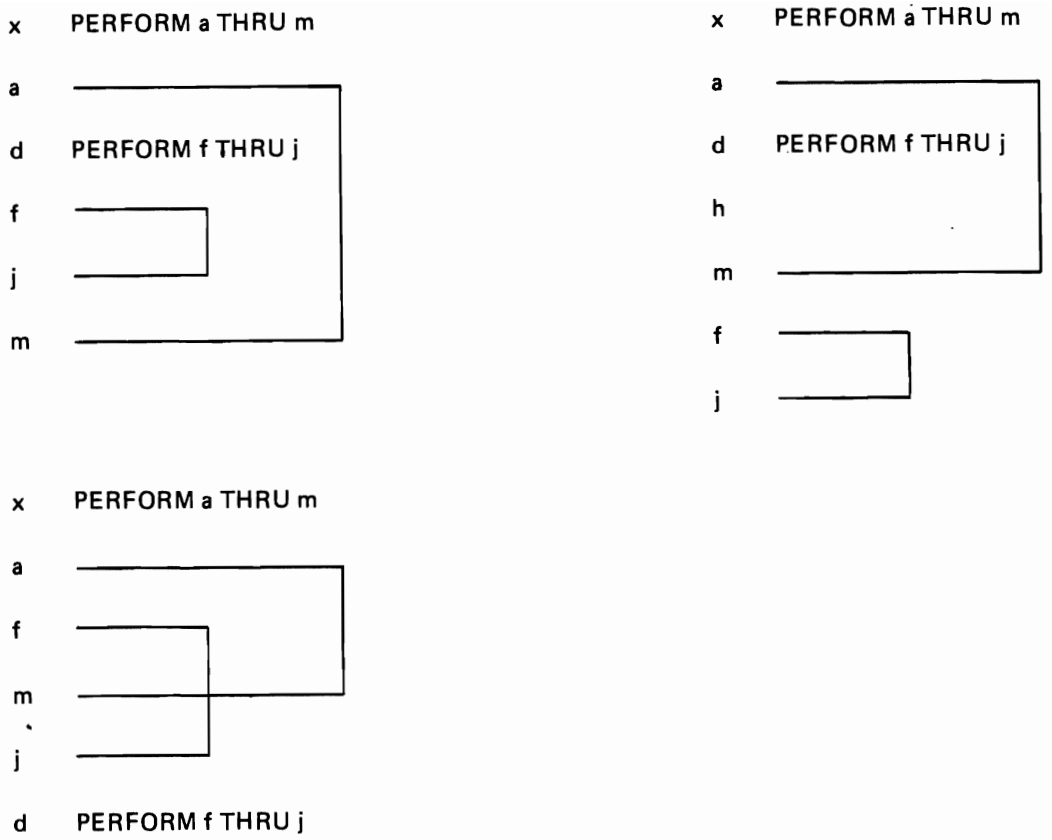


Figure 5-22. Valid PERFORM Statement Execution Sequences

The preceding rules refer to all four formats of the PERFORM statement. The following sections give rules applying to each individual format.

Format 1

Format 1 is the basic PERFORM statement. The procedure(s) referred to is executed once, and then control passes to the next executable statement following the PERFORM statement.

Format 2

Format 2 uses the TIMES option. Identifier-1 must name an integer item. The procedure(s) referred to is executed the number of times specified by the value in identifier-1 or integer-1. Control then passes to the next executable statement following the PERFORM statement. The following rules apply:

- If integer-1 or identifier-1 is zero or a negative number at the time the PERFORM statement is initiated, control passes to the statement following the PERFORM statement.

- After the PERFORM statement has been initiated, any reference to identifier-1 or change in the value of identifier-1 has no effect in varying the number of times the procedures are executed.

Format 3

Format 3 uses the UNTIL option. The procedure(s) referred to is performed until the condition specified by the UNTIL option is true. Control is then passed to the next executable statement following the PERFORM statement.

If condition-1 is true at the time the PERFORM statement is encountered, the specified procedure(s) is not executed.

Format 4

Format 4 uses the VARYING option. This option increments or decrements one or more identifiers or index-names according to the following rules. Once the condition(s) specified in the UNTIL option is satisfied, control is passed to the next executable statement following the PERFORM statement.

No matter how many variables are specified, the following rules apply:

- In the VARYING/AFTER options, when an index-name is specified:
  - The index-name is initialized and incremented or decremented according to the rules for the SET statement. For a description of the SET statement see *Table Handling* in Chapter 6.
  - In the associated FROM option, an identifier must be described as an integer and have a positive value; a literal must be a positive integer.
  - In the associated BY option, an identifier must be described as an integer; a literal must be a nonzero integer.

- In the FROM option, when an index-name is specified:
  - In the associated VARYING/AFTER option, an identifier must be described as an integer. It is initialized as described in the SET statement.
  - In the associated BY option, an identifier must be described as an integer and have a nonzero value; a literal must be a nonzero integer.

- In the BY option, identifiers and literals must have a nonzero value.

- Changing the values of identifiers and/or index-names in the VARYING, FROM, and BY options during execution changes the number of times the procedures are executed.

The way in which operands are incremented or decremented depends on the number of variables specified. In the following discussion, every reference to identifier-n refers equally to index-name-n except when identifier-n is the object of the BY option.

Varying One Identifier: The following actions take place:

1. Identifier-1 is set equal to its starting value, identifier-2 or literal-2.

2. Condition-1 is evaluated:
   a. If it is false, steps 3 through 5 are executed.
   b. If it is true, control passes directly to the statement following the PERFORM statement.

3. Procedure-1 through procedure-2 (if specified) are executed once.

4. Identifier-1 is augmented by identifier-3 (or literal-3), and condition-1 is evaluated again.

5. Steps 2 through 4 are repeated until condition-1 is true.

Figure 5-23 is a flowchart illustrating the logic of the PERFORM statement when one identifier is varied.

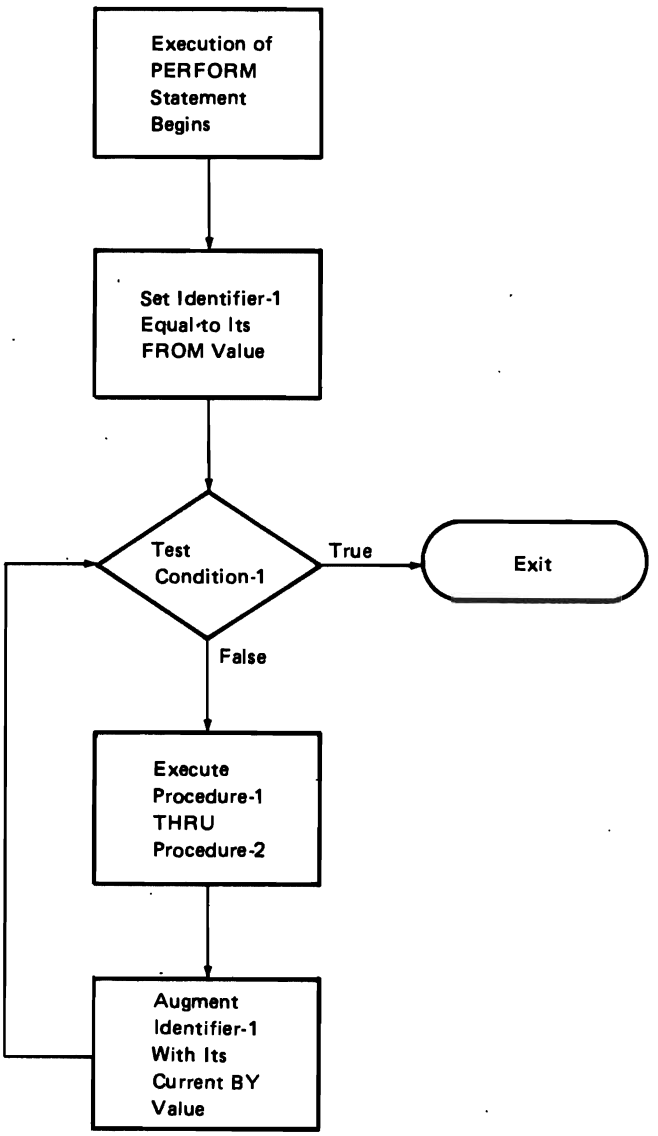


Figure 5-23. Format 4 PERFORM Statement Logic–Varying Two Identifiers

The following example shows a PERFORM statement varying one identifier. This PERFORM logic is executed 100 times.

```
 WORKING-STORAGE SECTION.
 77  SUB1 PIC 999.
 77  TOTAL-HOLD PIC 99 VALUE 57.
 77  HOLD-2 PIC 99 VALUE 10.
 77  HOLD-THE-SUM PIC 99 VALUE ZERO.
 01  TABLE-ELEMENT.
     03 ELEMENTS-OF-TABLE OCCURS 100 TIMES PIC 9.
 PROCEDURE DIVISION.
 100-START-PROCESSING.


*    THIS PERFORM LOGIC IS EXECUTED 100 TIMES.

     PERFORM SAMPLE-PERFORM THRU PERFORM-EXIT VARYING SUB1
     FROM 1 BY 1 UNTIL SUB1 > 100.


*    THIS ADD STATEMENT IS EXECUTED AFTER PERFORM IS DONE.

     ADD TOTAL-HOLD HOLD-2 GIVING HOLD-THE-SUM.


     DISPLAY 'TOTAL OF TWO VARIABLES = ' HOLD-THE-SUM.
     PERFORM ANOTHER-WAY-TO-INITIALIZE THRU AWTI-EXIT.
*    **************************.

*    THE TABLE WILL BE ALL ZEROS AND SHOULD PRINT AS SUCH.
*
*
*    **************************.
     DISPLAY '------------------THE-------TABLE----------'.
     DISPLAY TABLE-ELEMENT.
     STOP RUN.
 SAMPLE-PERFORM.
     MOVE ZEROS TO ELEMENTS-OF-TABLE (SUB1).


 PERFORM-EXIT. EXIT.
 ANOTHER-WAY-TO-INITIALIZE.
     MOVE ZEROS TO TABLE-ELEMENT.
 AWTI-EXIT. EXIT.
*    ******************END OF PROGRAM*****************************
```

Varying Two Identifiers: The following actions take place:

1. Identifier-1 and identifier-4 are set to their initial values, identifier-2 (or literal-2) and identifier-5 (or literal-5), respectively.

2. Condition-1 is evaluated:
   a. If it is false, steps 3 through 7 are executed.
   b. If it is true, control passes directly to the statement following the PERFORM statement.

3. Condition-2 is evaluated:
   a. If it is false, steps 4 through 6 are executed.
   b. If it is true, identifier-4 is set to the current value of identifier-5, and identifier-1 is augmented by identifier-3 (or literal-3), and step 2 is repeated.

4. Procedure-1 through procedure-2 (if specified) are executed once.

5. Identifier-4 is augmented by identifier-6 (or literal-6).

6. Steps 3 through 5 are repeated until condition-2 is true.

7. Steps 2 through 6 are repeated until condition-1 is true.

At the end of PERFORM statement execution, identifier-4 contains the current value of identifier-5. Identifier-1 has a value that exceeds the last used setting by the increment/decrement value (unless condition-1 was true at the beginning of PERFORM statement execution, in which case identifier-1 contains the current value of identifier-2).

Figure 5-24 is a flowchart illustrating the logic of the PERFORM statement when two identifiers are varied.

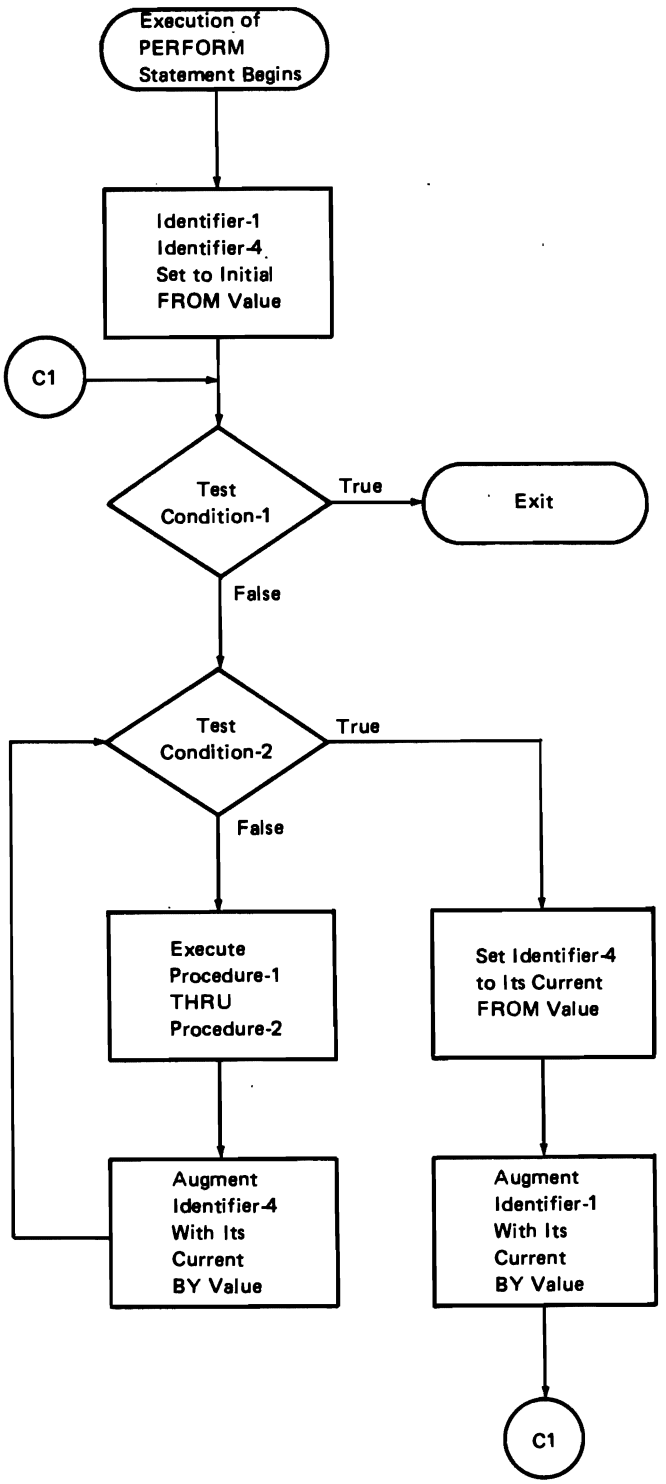


Figure 5-24. Format 4 PERFORM Statement Logic–Varying Three Identifiers

The following example shows a PERFORM statement varying two identifiers. This PERFORM logic is executed 126 times. This program searches a table and gives a total of female employees.

```
 DATA DIVISION.
 FILE SECTION.
 FD  EMPLOYEE-DATA
     BLOCK CONTAINS 1 RECORDS
     RECORD CONTAINS 80 CHARACTERS
     LABEL RECORDS STANDARD
     DATA RECORD IS EMPLOYEE-RECORD.
 01  EMPLOYEE-RECORD PIC X(80).
 WORKING-STORAGE SECTION.
 77  RECORDS-IN PIC 9(5) VALUE ZEROS.
 77  EOF-SW PIC X VALUE 'N'.
 01  HOLD-INPUT-RECORD.
     03  EMPLOYEE-SEX PIC 9.
         88  MALE VALUE IS 1.
         88 FEMALE VALUE IS 2.
     03 EMPLOYEE-RACE PIC 9.
         88 RACE-CODES VALUES ARE 1 THRU 7.
     03 EMPLOYEE-JOB-CLASS PIC 99.
         88 JOB-CLASS  VALUES ARE 01 THRU 18.
     03 FILLER PIC X(76) VALUE SPACES.
 01  EMPLOYEE-TABLE.
     03 E-SEX OCCURS 2 TIMES.
         05  E-RACE OCCURS 7 TIMES.
             07  E-JOB OCCURS  18 TIMES  PIC 99.
 01  SUB1 PIC 99.
 01  SUB2 PIC 99.
 01  SUB3 PIC 99.
 01  TOTAL-WOMEN PIC 9(5) VALUE ZEROS.
 PROCEDURE DIVISION.
 100-START-IT.
     OPEN INPUT EMPLOYEE-DATA OUTPUT PRINTED-REPORT.
     MOVE ZEROS TO EMPLOYEE-TABLE.
 200-READ-IT.
     READ EMPLOYEE-DATA RECORD INTO HOLD-INPUT-RECORD
     AT END MOVE 'Y' TO EOF-SW.
     ADD 1 TO RECORDS-IN.
 300-MAINLINE-LOGIC.
*    **************************************************.

*    THE PERFORM STATEMENT USING TWO VARIABLES WILL BE DONE 126 .
*    TIMES BY  THE COMPUTER.


*    **************************************************.

     PERFORM LOAD-TABLE UNTIL EOF-SW = 'Y'.


     PERFORM FIND-NUMBER-OF-WOMEN VARYING SUB2 FROM 1 BY 1
     UNTIL SUB2 > 7
     AFTER SUB3 FROM 1 BY 1 UNTIL SUB3 > 18.
```

```
     PERFORM WRITE-REPORT THRU WR-EXIT.
     DISPLAY 'TOTAL RECORDS IN ' RECORDS-IN.
     STOP RUN.
LOAD-TABLE.
     MOVE EMPLOYEE-SEX TO SUB1.
     MOVE EMPLOYEE-RACE TO SUB2.
     MOVE EMPLOYEE-JOB-CLASS TO SUB3.
     ADD 1 TO E-JOB (SUB1 SUB2 SUB3).
     PERFORM 200-READ-IT.
FIND-NUMBER-OF-WOMEN.
     ADD E-JOB (2 SUB2 SUB3) TO TOTAL-WOMEN.
WRITE-REPORT.
     MOVE TOTAL-WOMEN  TO PRINT-OUT.
     WRITE PRINT-OUT.
WR-EXIT. EXIT.
```

Varying Three Identifiers: The actions are the same as for varying two identifiers except that identifier-7 goes through the complete cycle each time that identifier-4 is augmented by identifier-6 or literal-6, which in turn goes through a complete cycle each time identifier-1 is varied.

At the end of PERFORM statement execution, identifier-4 and identifier-7 contain the current values of identifier-5 and identifier-8, respectively. Identifier-1 has a value exceeding its last used setting by one increment/decrement value (unless condition-1 was true at the beginning of PERFORM statement execution, in which case identifier-1 contains the current value of identifier-2).

Figure 5-25 is a flowchart illustrating the logic of the PERFORM statement when three identifiers are varied.

The following example shows a PERFORM statement varying three identifiers. This PERFORM logic is executed 250 times.

```
 WORKING-STORAGE SECTION.
 77  SUB1 PIC 99.
 77  SUB2 PIC 99.
 77  SUB3 PIC 99.
 77  TEST-IT PIC 99 VALUE 00.
 77  TOTAL-RECS PIC 99 VALUE ZEROS.
 01  COMPANY-TABLE.
     05 DIVISION-IN OCCURS 10 TIMES.
         10 DIVISION-NAME PIC X(10).
         10  DIVISION-NUMBER PIC 9(4).
             10 SECTION-IN OCCURS 5 TIMES.


                 15 UNIT-IN OCCURS 5 TIMES.
                    20 UNIT-NAME PIC X(5).
                    20 UNIT-NUMBER PIC 9(4).
 PROCEDURE DIVISION.
 100-START-PROCESSING.
*    ************************************************************.

*    THIS PERFORM LOGIC IS EXECUTED 250 TIMES BY THE COMPUTER.

*    ************************************************************.
     PERFORM ZERO-OUT-BIG-TABLE VARYING SUB1 FROM 1 BY 1
     UNTIL SUB1 > 10
*    SUB1 IS VARIED LAST BY THE COMPUTER.
     AFTER SUB2 FROM 1 BY 1 UNTIL SUB2 > 5
*    SUB2 IS VARIED  ******2ND****** BY THE COMPUTER.
     AFTER SUB3 FROM 1 BY 1 UNTIL SUB3 > 5.
*     ******SUB3 IS VARIED FIRST BY THE COMPUTER******* .
     PERFORM ADDRESS-THE-VARIABLES THRU ATV-EXIT.
     DISPLAY 'VARIABLE TEST-IT = ' TEST-IT.
     STOP RUN.
 ZERO-OUT-BIG-TABLE.
     MOVE ZEROS TO UNIT-IN (SUB1, SUB2, SUB3).
 ADDRESS-THE-VARIABLES.
     IF UNIT-NUMBER OF UNIT-IN OF SECTION-IN OF DIVISION-IN
     OF COMPANY-TABLE (3, 4, 5) = 0 ADD 1 TO TEST-IT.


 ATV-EXIT. EXIT.
```

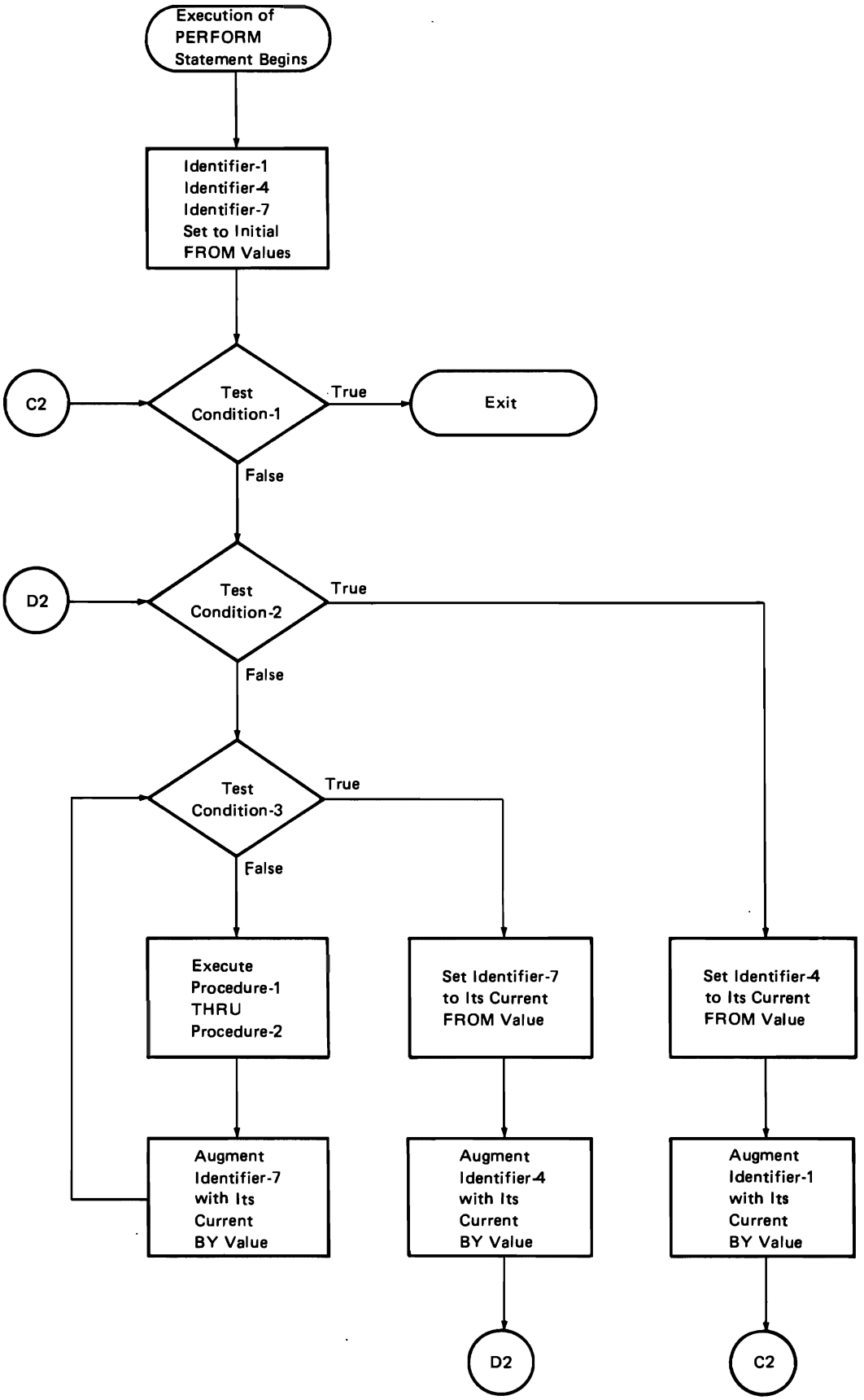


Figure 5-25. Format 4 PERFORM Statement Logic–Varying Three Identifiers

Programming Notes

The procedures executed by a PERFORM statement are in effect a closed subroutine that can be entered from other points in the program.

The Format 4 PERFORM statement is especially useful in table handling. One Format 4 PERFORM statement can serially search an entire 3-dimensional table.

Segmentation Information

A PERFORM statement appearing in a permanent segment can have in its range only one of the following:

- Sections, each of which has a segment number less than 50.
- Sections or paragraphs wholly contained in a single independent segment.

A PERFORM statement that appears in an independent segment can have in its range only one of the following:

- Sections, each of which has a segment number less than 50.
- Sections or paragraphs wholly contained within the same independent segment as the PERFORM statement.

Control is passed to the performed procedures only once for each execution of the PERFORM statement.

STOP Statement

The STOP statement halts the object program either temporarily or permanently.

Format

$$\underline{\text{STOP}} \left\{ \begin{matrix} \underline{\text{RUN}} \\ \text{literal} \end{matrix} \right\}$$

The literal can be numeric or nonnumeric, and can be any figurative constant except ALL literal. If the literal is numeric, it must be an unsigned integer.

When STOP literal is specified, the literal is communicated to the operator, and object program execution is suspended. Program execution is resumed only after operator intervention.

The action taken by the operator determines whether the job continues at the next executable statement in the sequence, the job step is canceled, or the entire job is canceled.

When STOP RUN is specified, execution of the object program is terminated, and control is returned to the system. If a STOP RUN statement appears in a sequence of imperative statements, it must be the last or the only statement in the sequence. All files should be closed before a STOP RUN statement is executed.

An implicit return to the calling program is always generated after the last statement in the source program. In a main program, this is equivalent to a STOP RUN. In a subprogram, this is equivalent to an EXIT PROGRAM.

For restrictions on the STOP RUN statement in calling and called programs, see *Program Termination Statements* in Chapter 8.

Programming Notes

The STOP literal statement is useful for special situations when operator intervention is needed during program execution.

COMPILER-DIRECTING STATEMENTS

Compiler-directing statements provide instructions to the COBOL compiler. The compiler-directing statements are COPY, ENTER, and USE.

Only the ENTER statement is discussed in this chapter. The COPY statement is discussed under *Source Program Library* in Chapter 6. The USE statements are discussed under *Debugging Features* in Chapter 6.

ENTER Statement

The System/34 COBOL compiler does not allow another source language to be used in COBOL source programs. Therefore, the ENTER statement is not required or used by the System/34 COBOL compiler.

Format

ENTER language-name [routine-name] .

If the ENTER statement is inserted in the source program, it is treated as a comment. Statements in the ENTERed language must not be included in the source program.

Chapter 6. Additional Functions

System/34 COBOL offers several additional functions that are useful to programmers who are writing more advanced applications. The additional functions provided by System/34 COBOL discussed in this chapter are:

- Table Handling
- SORT-MERGE
- Library Copy Facility
- Segmentation
- Inter-program Communication
- Debugging
- FIPS Flagger

Table Handling

Tables are often used in data processing. A table is a set of logically consecutive items, each of which has the same data description as the other items in the set. The items in a table can be described as separate contiguous items. However, this approach may not be satisfactory for two reasons. From a documentation standpoint, the homogeneity of the data items is not apparent; secondly, repetitive coding to reference unique data-names become a severe problem. Thus, a method of data reference is used which makes it possible to refer to all or to part of one table as an entity.

TABLE HANDLING CONCEPTS

In COBOL, a table is defined with an OCCURS clause in its data description. The OCCURS clause specifies that the named item is to be repeated as many times as stated. The item so named is considered a table element, and its name and description apply to each repetition (or occurrence) of the item. Because the occurrences are not given unique data-names, reference to a particular occurrence can be made only by specifying the data-name of the table element, together with the occurrence number of the desired item within the element.

The occurrence number is known as a subscript and the technique of supplying the occurrence number of individual table elements is called subscripting. A related technique, called indexing, is also available for table references. Both subscripting and indexing are described in the following sections.

Table Definition

COBOL allows tables in one, two or three dimensions.

To define a one-dimensional table, the user writes an OCCURS clause as part of the definition of a table element. However, the OCCURS clause must not appear in the description of a group item that contains the table element, that is, an OCCURS clause must not be specified for an 01-level item. For example:

```
01 TABLE-ONE.
   05 ELEMENT-ONE OCCURS 3 TIMES.
      10 ELEMENT-A PIC X(4).
      10 ELEMENT-B PIC 9(4).
```

TABLE-ONE is the group item that contains the table. ELEMENT-ONE names the table element of a one-dimensional table that occurs three times. ELEMENT-A and ELEMENT-B are elementary items subordinate to ELEMENT-ONE.

To define a two-dimensional table, a one-dimensional table is defined within each occurrence of another one-dimensional table.

For example:

```
01 TABLE-TWO.
   05 ELEMENT-ONE OCCURS 3 TIMES.
      10 ELEMENT-TWO OCCURS 3 TIMES.
         15 ELEMENT-A PIC X(4).
         15 ELEMENT-B PIC 9(4).
```

ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-TWO is an element of a two-dimensional table that occurs three times within each occurrence of ELEMENT-ONE. ELEMENT-A and ELEMENT-B are elementary items subordinate to ELEMENT-TWO.

To define a three-dimensional table, a one-dimensional table is defined within each occurrence of another one-dimensional table, which is itself contained within each occurrence of another one-dimensional table. For example:

```
01 TABLE-THREE.
   05 ELEMENT-ONE OCCURS 3 TIMES.
      10 ELEMENT-TWO OCCURS 3 TIMES.
         15 ELEMENT-THREE OCCURS 2 TIMES
            PICTURE X(8).
```

In this example, TABLE-THREE is the group item that contains the table. ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-TWO is an element of a two-dimensional table that occurs three times within each occurrence of ELEMENT-ONE. ELEMENT-THREE is an element of a three-dimensional table that occurs two times within each occurrence of ELEMENT-TWO. Figure 6-1 shows the storage layout for TABLE-THREE.

Table References

Whenever the user refers to a table element, or to any item associated with a table element the reference must indicate which occurrence is intended.

For a one-dimensional table, the occurrence number of the desired element gives the complete information. For tables of more than one dimension, an occurrence number for each dimension must be supplied. In the three-dimensional table defined in the previous discussion, for example, a reference to ELEMENT-THREE must supply the occurrence number for ELEMENT-ONE, ELEMENT-TWO, and ELEMENT-THREE. Either subscripting or indexing, described in the following paragraphs, can be used to supply the necessary references.

ELEMENT-ONE Occurs Three Times	ELEMENT-TWO Occurs Three Times	ELEMENT-THREE Occurs Two Times	Byte Displacement
ELEMENT-ONE (1)	ELEMENT-TWO (1, 1)	ELEMENT-THREE (1, 1, 1)	0
		ELEMENT-THREE (1, 1, 2)	8
	ELEMENT-TWO (1, 2)	ELEMENT-THREE (1, 2, 1)	16
		ELEMENT-THREE (1, 2, 2)	24
	ELEMENT-TWO (1, 3)	ELEMENT-THREE (1, 3, 1)	32
		ELEMENT-THREE (1, 3, 2)	40
ELEMENT-ONE (2)	ELEMENT-TWO (2, 1)	ELEMENT-THREE (2, 1, 1)	48
		ELEMENT-THREE (2, 1, 2)	56
	ELEMENT-TWO (2, 2)	ELEMENT-THREE (2, 2, 1)	64
		ELEMENT-THREE (2, 2, 2)	72
	ELEMENT-TWO (2, 3)	ELEMENT-THREE (2, 3, 1)	80
		ELEMENT-THREE (2, 3, 2)	88
ELEMENT-ONE (3)	ELEMENT-TWO (3, 1)	ELEMENT-THREE (3, 1, 1)	96
		ELEMENT-THREE (3, 1, 2)	104
	ELEMENT-TWO (3, 2)	ELEMENT-THREE (3, 2, 1)	112
		ELEMENT-THREE (3, 2, 2)	120
	ELEMENT-TWO (3, 3)	ELEMENT-THREE (3, 3, 1)	128
		ELEMENT-THREE (3, 3, 2)	136
			144

Figure 6.1 Storage Layout for TABLE-THREE

Subscripting

Subscripting is a method of providing table references through the use of subscripts. A subscript is an integer value that specifies the occurrence number of a table element. Subscripts can be used only when reference is made to an individual item within a table element.

Format

```
{data-name-1    }
{condition-name } [ { OF } data-name-2 ] ... ( subscript-1 [ , subscript-2 [ , subscript-3 ] ] )
                  { IN }
```

Data-name-1 must be the name of a table element, and can be qualified, if necessary. Note that when qualification is used, it is data-name-1 that is subscripted, not data-name-2.

The subscript can be represented either by a literal or a data-name.

A literal subscript must be an integer, and it must have a value of one or greater. The literal can have a positive sign or it may be unsigned. Negative subscript values are not permitted. For example, the following are valid literal subscript references to TABLE-THREE:

```
ELEMENT-THREE (1, 2, 1)

ELEMENT-THREE (2, 2, 1)
```

A data-name subscript must be described as an elementary numeric integer data item. A data-name subscript may be qualifed; it may not be subscripted or indexed. For example, assuming that SUB1, SUB2, and SUB3 are all items subordinate to SUBSCRIPT-ITEM, valid data-name subscript references to TABLE-THREE are:

```
ELEMENT-THREE (SUB1, SUB2, SUB3)

ELEMENT-THREE IN TABLE-THREE (SUB1 OF
   SUBSCRIPT-ITEM, SUB2 OF SUBSCRIPT-ITEM,
   SUB3 OF SUBSCRIPT-ITEM)
```

The set of one to three subscripts must be written within a balanced pair of parentheses immediately following data-name-1 or its last qualifier. One or more spaces may optionally precede the opening parenthesis.

When more than one subscript is specified, each subscript must be separated from the next by either a space or a comma and a space.

When more than one subscript is required, the subscripts are written in the order of successively less inclusive data dimensions. For example, in the table reference ELEMENT-THREE (3, 2, 1), the first value (3) refers to the occurrence within ELEMENT-ONE, the second value (2) refers to the occurrence within ELEMENT-TWO, and the third value (1) refers to the occurrence within ELEMENT-THREE.

The lowest possible subscript value is 1; this value points to the first occurrence within the table element. The next sequential elements are pointed to by subscripts with values 2, 3, and so on. The highest permissible subscript value in any particular table element is the maximum number of occurrences specified in the OCCURS clause. In the example in the above paragraph, the highest possible subscript value for ELEMENT-ONE is 3, for ELEMENT-TWO is 3, and for ELEMENT-THREE is 2.

The following example shows subscripting using a 3-level table. In this example, UNIT-NUMBER could also be referenced as UNIT-NUMBER (3, 4, 5).

```
WORKING-STORAGE SECTION.
77  SUB1 PIC 99.
77  SUB2 PIC 99.
77  SUB3 PIC 99.
77  TEST-IT PIC 99 VALUE 00.
77  TOTAL-RECS PIC 99 VALUE ZEROS.
01  COMPANY-TABLE.
    05 DIVISION-IN OCCURS 10 TIMES.
        10 DIVISION-NAME PIC X(10).
        10  DIVISION-NUMBER PIC 9(4).
                10 SECTION-IN OCCURS 5 TIMES.


                                15 UNIT-IN OCCURS 5 TIMES.
                                20 UNIT-NAME PIC X(5).
                            20 UNIT-NUMBER PIC 9(4).
PROCEDURE DIVISION.
100-START-PROCESSING.


    PERFORM ZERO-OUT-BIG-TABLE VARYING SUB1 FROM 1 BY 1
    UNTIL SUB1 > 10
*   SUB1 IS VARIED LAST BY THE COMPUTER.
    AFTER SUB2 FROM 1 BY 1 UNTIL SUB2 > 5
*   SUB2 IS VARIED  ******2ND****** BY THE COMPUTER.
    AFTER SUB3 FROM 1 BY.1 UNTIL SUB3 > 5.
*    ******SUB3 IS VARIED FIRST BY THE COMPUTER******* .
    PERFORM ADDRESS-THE-VARIABLES THRU ATV-EXIT.
    DISPLAY 'VARIABLE TEST-IT = ' TEST-IT.
    STOP RUN.
ZERO-OUT-BIG-TABLE.
    MOVE ZEROS TO UNIT-IN (SUB1, SUB2, SUB3).
ADDRESS-THE-VARIABLES.
    IF UNIT-NUMBER OF UNIT-IN OF SECTION-IN OF DIVISION-IN
    OF COMPANY-TABLE (3, 4, 5) = 0 ADD 1 TO TEST-IT.


ATV-EXIT. EXIT.
```

Subscripting

Indexing

Indexing is the method of providing table references through the use of indexes. An index is a compiler-generated storage area used to store table element occurrence numbers. The index contains a value that corresponds to an occurence number.

Format

```
{ data-name      } ( { index-name-1 [{±} literal-2] }
{ condition-name }   { literal-1                    }

[ , { index-name-2 [{±} literal-4] } [ , { index-name-3 [{±} literal-6] } ] ] )
  { literal-3                      }     { literal-5                    }
```

Each index-name identifies an index to be used in table references. The index-name is specified through the OCCURS clause.

Each index named is a compiler-generated storage area, 2 bytes in length. Two forms of indexing are provided: direct and relative.

In direct indexing, the index-name is in the form of a subscript. In relative indexing, the index-name is followed by a space, a + or a -, another space, and an unsigned numeric literal. The literal is considered to be an occurrence number, and is converted to an index value before being added to or subtracted from the index-name index.

To be valid during execution, an index value must correspond to a table element occurrence not less than one, or greater than the highest permissible occurrence number. This restriction applies to both direct and relative indexing.

An index-name must be initialized through a SET, PERFORM-Format 4, or SEARCH ALL statement before it is used in a table reference.

One or more index references (direct or relative) can be specified together with literal subscripts.

Further information on index-names is given later in this chapter in the description of the INDEXED BY option of the OCCURS clause.

Restrictions on Subscripting and Indexing

- A data-name must not be subscripted or indexed when it is being used as a subscript or qualifier.
- Indexing is not permitted when subscripting is not permitted.
- An index can be modified only by a PERFORM, SEARCH, or SET statement.
- When a literal is used in a subscript, it must be a positive or unsigned integer.
- When a literal is used in relative indexing, it must be an unsigned integer.

Note: If the value of the index or subscript is outside the range specified in the OCCURS clause, unpredictable results or abnormal termination may occur. To avoid this problem, include the following sentence when referencing tables:

```
IF field GREATER THAN occurs-integer OR field
LESS THAN 1, PERFORM error-routine, GO TO
END-OF-JOB.
```

Field is the subscript or index being used, and *occurs-integer* is the integer in the OCCURS clause of the table level being referenced.

Table Initialization

A table can contain static values or dynamic values. Static values remain the same through every execution of the object program. When this is true, the initial values of table elements can be specified in Working-Storage in one of two ways:

- The table can be described as a record containing contiguous subordinate data description entries, each of which contains a VALUE clause for the initial value. The record is then redescribed through a REDEFINES entry that contains a subordinate entry with an OCCURS clause. Because of the OCCURS clause, the subordinate entries of the redefined entry are repeated. For example:

```
01 TABLE-ONE.
    05 ELEMENT-ONE PICTURE X VALUE '1'.
    05 ELEMENT-TWO PICTURE X VALUE '2'.
    05 ELEMENT-THREE PICTURE X VALUE '3'.
    05 ELEMENT-FOUR PICTURE X VALUE '4'.
01 TABLE-TWO REDEFINES TABLE-ONE.
    05 OCCURS-ELEMENT OCCURS 4 TIMES
       PICTURE X.
```

- If the subordinate entries do not require separate handling, the VALUE of the entire entry can be given in the entry that names the table. The lower level entries then contain OCCURS clauses, and show the hierarchical structure of the table. The subordinate entries must not contain VALUE clauses. For example:

```
01  TABLE-ONE VALUE '1234'.
    05 TABLE-TWO OCCURS 4 TIMES
       PICTURE X.
```

Dynamic values may change during one execution of the object program, or from one execution to another. If the dynamic values are always the same at the beginning of object program execution, they can be initialized in the same manner as static values. If the initial values change from one execution to the next, then the table can be defined without initial values, and the changed values can be placed in the table before any table reference is made.

The following example shows two ways of initializing a table with zeros.

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  SUB1 PIC 999.
       77  TOTAL-HOLD PIC 99 VALUE 57.
       77  HOLD-2 PIC 99 VALUE 10.
       77  HOLD-THE-SUM PIC 99 VALUE ZERO.
       01  TABLE-ELEMENT.
           03 ELEMENTS-OF-TABLE OCCURS 100 TIMES PIC 9.

       PROCEDURE DIVISION.
       100-START-PROCESSING.
           PERFORM SAMPLE-PERFORM THRU PERFORM-EXIT VARYING SUB1
           FROM 1 BY 1 UNTIL SUB1 > 100.
           ADD TOTAL-HOLD HOLD-2 GIVING HOLD-THE-SUM.
      *    THIS ADD STATEMENT IS EXECUTED AFTER THE PERFORM IS DONE.
           DISPLAY 'TOTAL OF TWO VARIABLES = ' HOLD-THE-SUM.
           PERFORM ANOTHER-WAY-TO-INITIALIZE THRU AWTI-EXIT.
      *    **************************.
      *    THE TABLE WILL BE ALL ZEROS AND SHOULD PRINT AS SUCH.
      *
      *
      *    **************************.
           DISPLAY '--------------------THE-------TABLE----------'.
           DISPLAY TABLE-ELEMENT.
           STOP RUN.
       SAMPLE-PERFORM.
           MOVE ZEROS TO ELEMENTS-OF-TABLE (SUB1).

       PERFORM-EXIT. EXIT.
       ANOTHER-WAY-TO-INITIALIZE.
           MOVE ZEROS TO TABLE-ELEMENT.
       AWTI-EXIT. EXIT.
```

Initializing Table To Zeros

DATA DIVISION–TABLE HANDLING

COBOL Data Division clauses used for Table Handling are the OCCURS clause and the USAGE IS INDEX clause.

OCCURS Clause

The OCCURS clause eliminates the need to specify separate entries for repeated data items; it also supplies the information necessary for the use of subscripts or indexes. The formats of the OCCURS clause are as follows:

Format

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

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

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

The subject of an OCCURS clause is the data-name of the data item containing the OCCURS clause. Except for the OCCURS clause itself, data description clauses used with the subject apply to each occurrence of the item described.

Whenever the subject is referred to in a statement other than a SEARCH statement or is the object of a REDEFINES clause, the subject must be subscripted or indexed. When it is subscripted or indexed, the subject refers to one occurrence within the table element.

Whenever the subject is referred to in a SEARCH statement or is the object of a REDEFINES clause, the subject must not be subscripted or indexed. When the subject is not subscripted or indexed, it represents the entire table.

The table must contain less than 32768 occurrences, and the length of the table must be less than 32768 bytes.

All data-names used in the OCCURS clause may be qualified; they may not be subscripted or indexed.

All integers must be positive nonzero integers.

The OCCURS clause cannot be specified in a data description entry that:

- Has a level-01, level-66, level-77, or level-88 number.
- Describes an item of variable size (an item is of variable size if any subordinate entry contains an OCCURS DEPENDING ON clause).
- Describes redefined data items. (However, a redefined item can be subordinate to an item containing an OCCURS clause.) See the *REDEFINES Clause* in Chapter 4.

Fixed Length Tables

When an OCCURS DEPENDING ON clause is not used, integer-2 specifies the exact number of occurrences.

Integer-2 must be greater than zero and less than 32768.

Because three subscripts or indexes are allowed, three nested levels of this format of the OCCURS clause are allowed.

Variable Length Tables

When the OCCURS DEPENDING ON clause is specified, integer-1 represents the minimum number of occurrences, and integer-2 represents the maximum number of occurrences. The value of integer-1 must be one or greater; it must also be less than integer-2. Integer-2 must be less than 32768. The length of the subject item is fixed; it is only the number of repetitions of the subject item that is variable.

Data-name-1 specifies the object of the OCCURS DEPENDING ON clause. The object is the data item whose current value represents the current number of occurrences of the subject item. The object of the OCCURS DEPENDING ON clause:

- Must be described as a positive integer. That is, if data-name-1 is described as a signed item, then at execution time it must contain positive data.
- Must not occupy any storage position within the range of this table. That is, the object must not occupy any storage position from the first character position in this table through the last character position in this record description entry.
- Must contain a value within the range of integer-1 and integer-2, inclusive.

The value of the object of the OCCURS DEPENDING ON clause specifies that part of the table element available to the object program. Items whose occurrence numbers exceed the value of the object are not available. If, during execution, the value of the object is reduced, the contents of items whose occurrence numbers exceed the new value of the object are unpredictable.

When a group item containing a subordinate OCCURS DEPENDING ON item is referred to, the current value of the object determines which part of the table area is used in the operation.

In one record description entry, any entry that contains an OCCURS DEPENDING ON clause may be followed only by items subordinate to it. The OCCURS DEPENDING ON clause cannot be specified as subordinate to another OCCURS clause. However, the Format 1 OCCURS clause may be specified as subordinate to the OCCURS DEPENDING ON clause; in this case, a table of up to three dimensions may be specified.

ASCENDING/DESCENDING KEY Option

The ASCENDING/DESCENDING KEY option specifies that the repeated data is arranged in ascending or descending order according to the values contain in data-name-2, data-name-3, and so on. The data-names are listed in their descending order of significance. The ASCENDING/DESCENDING KEY data items are used by the SEARCH ALL statement for a search of the table element.

The order is determined by the rules for comparison of operands. (See *Relation Condition* in Chapter 5.)

Data-name-2 must be the name of the subject entry or the name of an entry subordinate to the subject entry. If data-name-2 names the subject entry, that entire entry becomes an ASCENDING/DESCENDING KEY. If data-name-2 does not name the subject entry, then data-name-2, data-name-3, and so on:

- Must be subordinate to the subject of the table entry itself.
- Must not be subordinate to any other entry that contains an OCCURS clause.
- Must not themselves contain an OCCURS clause.

The following example illustrates the specification of ASCENDING/DESCENDING KEY data items:

```
WORKING-STORAGE SECTION.
77 CURRENT-WEEK PICTURE 99.
01 TABLE-RECORD.
   05 EMPLOYEE-TABLE OCCURS 100 TIMES
      ASCENDING KEY IS WAGE-RATE
      EMPLOYEE-NO INDEXED BY A, B.
      10 EMPLOYEE-NAME PIC X(20).
      10 EMPLOYEE-NO PIC 9(6).
      10 WAGE-RATE PIC 9999V99.
      10 WEEK-RECORD OCCURS 52 TIMES
         ASCENDING KEY IS WEEK-NO
         INDEXED BY C.
         15 WEEK-NO PIC 99.
         15 AUTHORIZED-ABSENCES PIC 9.
         15 UNAUTHORIZED-ABSENCES PIC 9.
         15 LATENESSES PIC 9.
```

The keys for EMPLOYEE-TABLE are subordinate to that entry, and the key for WEEK-RECORD is subordinate to that subordinate entry.

When the ASCENDING/DESCENDING KEY option is specified, the following rules apply:

- Keys must be listed in decreasing order of significance
- The programmer is responsible for ensuring that the data present in the table is arranged in ascending/descending key sequence according to the collating sequence in use

In the preceding example, records in EMPLOYEE-TABLE must be arranged in ascending order of WAGE-RATE and in ascending order of EMPLOYEE-NO within WAGE-RATE. Records in WEEK-RECORD must be arranged in ascending order of WEEK-NO. If they are not, SEARCH ALL statement results will be unpredictable.

INDEXED BY Option

The INDEXED BY option specifies the indexes that can be used with this table element. The INDEXED BY option is required if indexing is used to refer to this table element.

Each index-name must follow the rules for formation of a user-defined word; at least one character must be alphabetic. Each index-name specifies an index to be created by the compiler for use by the program. These index-names are not data-names and are not identified elsewhere in the COBOL program; instead, they can be regarded as private special registers for the use of this object program only. Therefore, they are not data or part of any data hierarchy; as such, each must be unique. An INDEX-NAME can only be referenced by a PERFORM, SET, or SEARCH statement, as a parameter in the USING phrase in a CALL statement, or in a relational condition comparison.

USAGE IS INDEX Clause

The USAGE IS INDEX clause specifies that the data item named has an index format. Such an item is an index data item.

Format

```
[ USAGE IS ] INDEX
```

An index data item is an elementary item that can be used to save index-name values for future reference. Through the SET statement, an index data item can be assigned an index-name value. An index value corresponds to the displacement for an occurrence number in the table, that is (occurrence-number – 1) * entry length.

An index data item can be referred to directly only in a SEARCH statement, a SET statement, a relation condition, the USING option of the Procedure Division header, or the USING option of the CALL statement. An index data item can be part of a group item referred to in a MOVE statement or an input/output statement.

An index data item saves values equivalent to table occurrences; however, it is not itself necessarily defined as part of any table. Thus, when it is referenced directly in a SEARCH or SET statement, or indirectly in a MOVE or input/output statement, there is no conversion of values when the statement is executed.

The USAGE IS INDEX clause can be written at any level. If a group item is described with the USAGE IS INDEX clause, it is the elementary items within the group that are index data items; the group itself is not an index data item, and the group name cannot be used in SEARCH and SET statements or in relation conditions. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

An index data item cannot be a conditional variable; it cannot have a subordinate level-88.

The SYNCHRONIZED, JUSTIFIED, PICTURE, BLANK WHEN ZERO, or VALUE clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

PROCEDURE DIVISION–TABLE HANDLING

In the Procedure Division, the SEARCH and SET statements can be specified with indexed tables. There are also special rules involving table handling elements when they are used in relation conditions.

Relation Conditions

Comparisons involving index-names or index data items conform to the following rules:

- The comparison of two index-names is actually the comparison of the corresponding occurrence numbers.
- In the comparison of an index-name with a data item (other than an index data item) or in the comparison of an index-name with a literal, the occurrence number that corresponds to the value of the index-name is compared with the data item or literal.
- In the comparison of an index data item with an index-name or another index data item, the actual values are compared without conversion. Results of any other comparison involving an index data item are undefined.

Figure 6-2 shows permissible comparisons for index-names and index data items.

First Operand \ Second Operand	Index-name[1]	Index Data Item[2]	Data-name (numeric integer only)	Numeric Literal (integer only)
Index-name[1]	Compare occurrence number	Compare without conversion	Compare occurrence number with data-name	Compare occurrence number with literal
Index Data Item[2]	Compare without conversion	Compare without conversion	Invalid	Invalid
Data-name (numeric integer only)	Compare occurrence number with data-name	Invalid		
Numeric Literal (integer only)	Compare occurrence number with literal	Invalid		

[1]See *OCCURS Clause* earlier in this chapter.
[2]See *USAGE IS INDEX Clause* earlier in this chapter.

Figure 6-2. Permissible Comparisons for Index-Names and Index Data Items

SEARCH Statement

The SEARCH statement searches a table for an element that satisfies the specified condition, and adjusts the associated index to indicate that element. The formats for the SEARCH statement are:

Format 1

```
SEARCH identifier-1 [ VARYING { identifier-2   } ]  [ AT END imperative-statement-1 ]
                              { index-name-1   }

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

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

Format 2

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

    WHEN { data-name-1 { IS EQUAL TO } { identifier-3 }   }
         {             { IS =       } { literal-1    }   }
         {                            { arithmetic-expression-1 } }
         { condition-name-1                               }

    [ [ AND { data-name-2 { IS EQUAL TO } { identifier-4 }   } ] ] ...
            {             { IS =       } { literal-2    }   }
            {                            { arithmetic-expression-2 } }
            { condition-name-2                               }

    { imperative-statement-2 }
    { NEXT SENTENCE          }
```

The Data Division description of identifier-1 must contain an OCCURS clause with the INDEXED BY option.

When specified in the SEARCH statement, identifier-1 must refer to all occurrences within the table element; it must not be subscripted or indexed.

Identifier-1 can be a data item subordinate to a data item that contains an OCCURS clause; it can be a part of a two- or three-dimensional table. In this case, the data description entry must specify an INDEXED BY option for each dimension of the table.

SEARCH statement execution modifies only the value in the index-name associated with identifier-1 (and, if present, of index-name 1 or identifier-2). Therefore, to search an entire two- or three-dimensional table, a SEARCH statement must be executed for each dimension. Before each execution, SET statements must be executed to reinitialize the associated index-names.

In the AT END and WHEN options, control passes to the next sentence after the imperative-statement is executed if any of the specified imperative-statements do not end with a GO TO statement.

Format 1

Format 1 SEARCH statement execution causes a serial search to be executed, beginning at the current index setting.

If the value of the index-name associated with identifier-1 is not greater than the highest possible occurrence number, when the search begins the following actions take place:

1. The condition(s) in the WHEN option are evaluated in the order they are written.

2. If none of the conditions are satisfied, the index-name for identifier-1 is incremented to correspond to the next table element, and step 1 is repeated.

3. If upon evaluation, one of the WHEN conditions is satisfied, the search terminates immediately, and the imperative-statement associated with that condition is executed. The index-name points to the table element that satisfied the condition.

4. If the end of the table is reached (that is, the incremented index-name value is greater than the highest possible occurrence number) without the WHEN condition being satisfied, the search terminates as described in the next paragraph.

If, when the search begins, the value of the index-name associated with identifier-1 is greater than the highest possible occurrence number, the search immediately ends, and, if specified, the AT END imperative-statement is executed. If the AT END option is omitted, control passes to the next sentence.

Each WHEN option condition may be any condition as described under *Conditional Expressions* in Chapter 5.

VARYING Index-Name-1 Option: When the VARYING index-name-1 option is omitted, the first (or only) index-name for identifier-1 is used for the search. When the VARYING index-name-1 option is specified, one of the following actions takes place:

- If index-name-1 is an index for identifier-1, this index is used for the search. Otherwise, the first (or only) index-name is used.

- If index-name-1 is an index for another table element, then the first (or only) index-name for identifier-1 is used for the search; the occurrence number represented by index-name-1 is incremented by the same amount as the search index-name and at the same time.

VARYING Identifier-2 Option: When this option is specified, the first (or only) index-name for identifier-1 is used for the search.

Identifier-2 must be either an index data item or an elementary integer item. During the search, one of the following actions takes place:

- If identifier-2 is an index data item, then whenever the search index is incremented, the specified index item is simultaneously incremented by the same amount.

- If identifier-2 is an integer data item, then whenever the search index is incremented, the specified data item is simultaneously incremented by 1.

Figure 6-3 is a flowchart of a Format 1 SEARCH operation containing two WHEN Options.

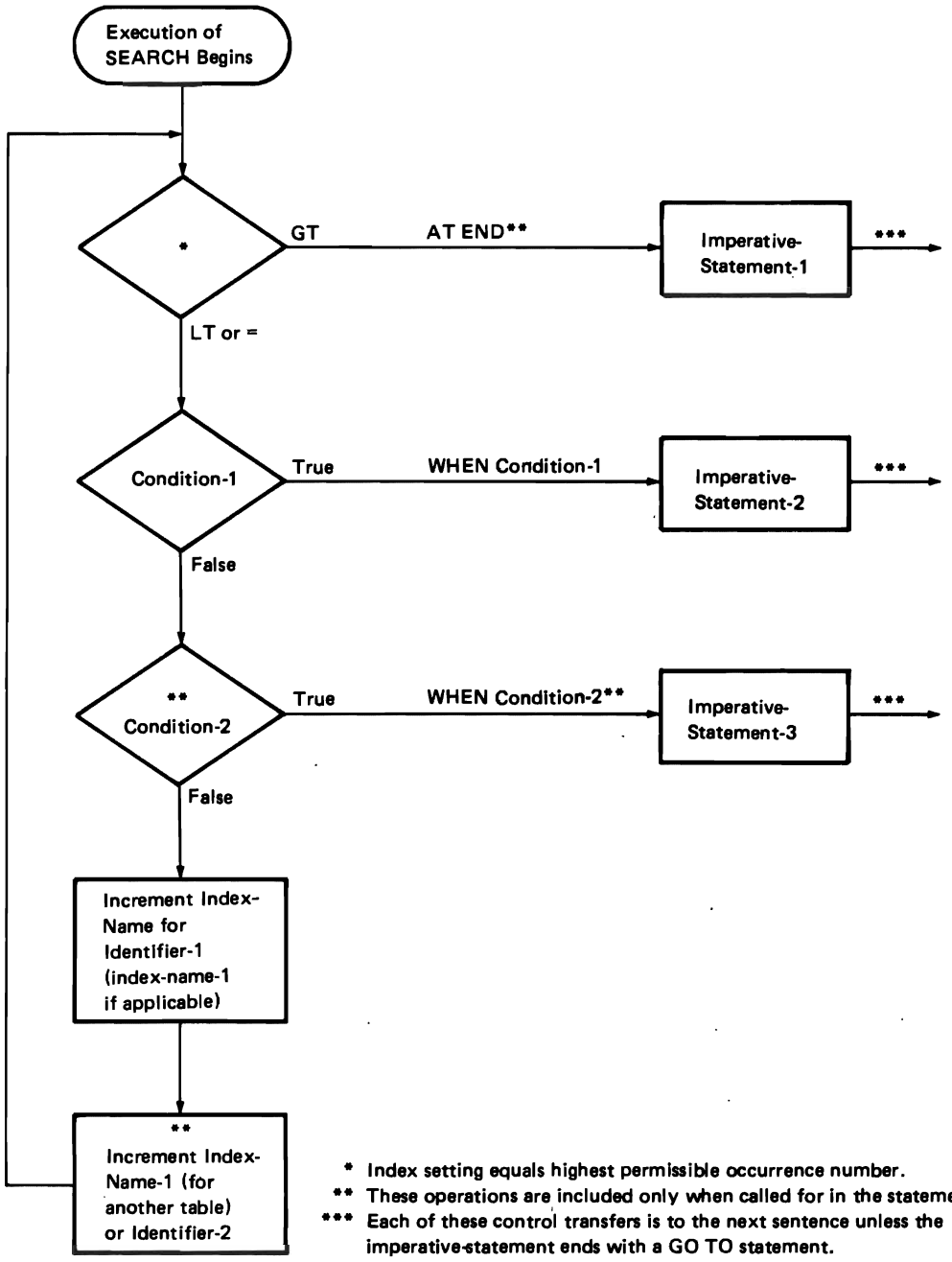


Figure 6-3. Format 1 SEARCH with Two WHEN options

Format 2

Format 2 SEARCH ALL statement execution causes a serial search to be executed, beginning with the first element of the table. The search index need not be initialized by SET statements, because its setting is varied during the search operation. The index used is always the index that is associated with the first index-name specified in the OCCURS clause.

If the WHEN option cannot be satisfied for any setting of the index within this range, the search is unsuccessful. If the AT END option is specified, the AT END imperative-statement is executed, but if the AT END option is not specified, control is passed to the next sentence; in either case, the final setting of the index is not predictable.

Note: When such undefined index values are used, error SYS–0014 could be encountered.

If the WHEN option can be satisfied, control passes to imperative-statement-2 and the index contains a value indicating an occurrence that allows the WHEN condition(s) to be satisfied.

WHEN Condition-Name Option: If the WHEN condition-name option is specified, each condition-name specified must have only a single value, and each must be associated with an ASCENDING/DESCENDING KEY identifier for this table element.

WHEN Relation-Condition Option: If WHEN relation-condition is specified, the following considerations apply:

- Data-name-1 or data-name-2 must specify an ASCENDING/DESCENDING KEY data item in the identifier-1 table element and must be indexed by the first identifier-1 index-name, along with other indexes or literals as required. Each data-name can be qualified.
- Identifier-3 and identifier-4 must not be an ASCENDING/DESCENDING KEY data item for identifier-1 or an item that is indexed by the first index-name for identifier-1.
- Literal-1 or literal-2 must be a positive or unsigned numeric integer.
- Arithmetic-expression-1 or arithmetic-expression-2 can be any of those defined under *Arithmetic Expressions* in Chapter 5 with the following restriction: any identifier in the arithmetic-expression must not be an ASCENDING/DESCENDING KEY data item for identifier-1 or an item that is indexed by the first index-name for identifier-1.
- When an ASCENDING/DESCENDING KEY data item is specified either explicitly or implicitly in the WHEN option, then all preceding ASCENDING/DESCENDING KEY data-names for identifier-1 must also be specified.

The results of a SEARCH ALL operation are predictable only when both of the following apply:

- The data in the table is ordered in ascending/descending key sequence
- The contents of the ASCENDING/DESCENDING keys specified in the WHEN clause provide a unique table reference

Programming Notes

Index data items cannot be used as subscripts or indexes, because of the restrictions on direct reference to them. The use of a direct indexing reference together with a relative indexing reference for the same index-name allows reference to two different occurrences of a table element for comparison purposes.

When the object of the VARYING option is an index-name for another table element, one Format 1 SEARCH statement looks at two table elements at once.

One Format 4 PERFORM statement can search an entire multidimensional table.

To ensure correct execution of a PERFORM or SEARCH statement for a variable length table, the programmer must make sure that the object of the OCCURS DEPENDING ON clause (data-name-1) contains a value that correctly specifies the current length of the table.

SEARCH Example

The following example searches an inventory table for items that match those from input data. The key is INVENTORY-NUMBER.

```
DATA DIVISION.
FILE SECTION.
FD  SALES-DATA
    BLOCK CONTAINS 1 RECORDS
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS STANDARD
    DATA RECORD IS SALES-REPORTS.
01  SALES-REPORTS PIC X(80).
FD  PRINTED-REPORT
    BLOCK CONTAINS 1 RECORDS
    RECORD CONTAINS 132 CHARACTERS
    LABEL RECORDS OMITTED
    DATA RECORD IS PRINTER-OUTPUT.
01  PRINTER-OUTPUT PIC X(132).
FD  INVENTORY-DATA
    BLOCK CONTAINS 1 RECORDS
    RECORD CONTAINS 40 CHARACTERS
    LABEL RECORDS STANDARD
    DATA RECORD IS INVENTORY-RECORD.
01  INVENTORY-RECORD.
    03  I-NUMBER       PIC 9(4).
    03 INV-ID          PIC X(26).
    03  I-COST         PIC 9(8)V99.
WORKING-STORAGE SECTION.
77  EOF-SW PIC X VALUE 'N'.
77  EOF-SW2  PIC X VALUE 'N'.
77  SUB1  PIC 99.
77  RECORDS-NOT-FOUND PIC 9(5) VALUE ZEROS.
77  TOTAL-COSTS PIC 9(10) VALUE ZEROS.
01  HOLD-INPUT-DATA.
    03  INVENTORY-NUMBER PIC 9999.
    03  PURCHASE-COST PIC 9(4)V99.
    03  PURCHASE-DATE PIC 9(6).
    03  FILLER PIC X(64).
01  PRINTER-SPECS.
    03  PRINT-LINE.
        05  OUTPUT-ITEM-NUMBER PIC ZZZ9.
        05 FILLER PIC X(48) VALUE SPACES.
        05  TOTAL-COSTS-O  PIC $(8).99.
01  PRODUCT-TABLE.
    05  INVENTORY-NUMBERS OCCURS 50 TIMES
        ASCENDING KEY ITEM-NUMBER
        INDEXED BY INDEX-1.
             07  ITEM-NUMBER PIC 9(4).
             07 ITEM-DESCRIPTION PIC X(26).
             07  ITEM-COST PIC 9(8)V99.
```

SEARCH Example (Continued)

```
PROCEDURE DIVISION.
100-START-IT.
    OPEN INPUT SALES-DATA INVENTORY-DATA OUTPUT PRINTED-REPORT.
    MOVE HIGH-VALUES TO PRODUCT-TABLE.
READ-INVENTORY-DATA.
    READ INVENTORY-DATA AT END MOVE 'Y' TO EOF-SW2.
LOAD-TABLE-ROUTINE.
    PERFORM LOAD-IT VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 > 50.
110-READ-IT.
    READ SALES-DATA INTO HOLD-INPUT-DATA AT END

    MOVE 'Y' TO EOF-SW.
200-MAIN-ROUTINE.
    PERFORM PROCESS-DATA UNTIL EOF-SW = 'Y'.
    MOVE TOTAL-COSTS TO TOTAL-COSTS-O.
    PERFORM WRITE-REPORT THRU WRITE-REPORT-EXIT.
    DISPLAY 'RECORDS NOT FOUND = ' RECORDS-NOT-FOUND.
    STOP RUN.

PROCESS-DATA.
    SEARCH ALL INVENTORY-NUMBERS AT END
    PERFORM KEY-NOT-FOUND THRU NOT-FOUND-EXIT
    WHEN INVENTORY-NUMBER IS = ITEM-NUMBER (INDEX-1)
            MOVE ITEM-NUMBER (INDEX-1) TO OUTPUT-ITEM-NUMBER
        MOVE ITEM-COST (INDEX-1) TO TOTAL-COSTS-O
            ADD ITEM-COST (INDEX-1) TO TOTAL-COSTS.
    PERFORM WRITE-REPORT THRU WRITE-REPORT-EXIT.
    PERFORM 110-READ-IT.
KEY-NOT-FOUND.
    ADD 1 TO RECORDS-NOT-FOUND.
NOT-FOUND-EXIT. EXIT.
LOAD-IT.
    MOVE INVENTORY-RECORD TO INVENTORY-NUMBERS (SUB1).
    PERFORM READ-INVENTORY-DATA.

WRITE-REPORT.
    WRITE PRINTER-OUTPUT FROM PRINTER-SPECS.
WRITE-REPORT-EXIT. EXIT.
*    *********END SAMPLE SEARCH PROGRAM************************.
```

SET Statement

The SET statement can:

- Establish reference points for table handling operations by setting index-names to values associated with table elements
- Transfer values between index-names and other elementary data items
- Alter the status of external UPSI switches
- Alter the value of conditional variables

The formats of the SET statement are:

Format 1

```
SET { identifier-1   [ ,identifier-2 ]  . . . }  TO { identifier-3 }
    { index-name-1   [ ,index-name-2 ]  . . . }     { index-name-3 }
                                                    { integer-1    }
```

Format 2

```
SET index-name-4 [ ,index-name-5 ] . . . { UP BY   } { identifier-4 }
                                         { DOWN BY }  { integer-2   }
```

Format 3

```
SET mnemonic-name-1 [ ,mnemonic-name-2 ] . . . TO { ON  }
                                                  { OFF }
```

Format 4

```
SET condition-name-1 [ ,condition-name-2 ] . . . TO TRUE
```

Index-names are related to a given table through the INDEXED BY option of the OCCURS clause. Index-names specified in the INDEXED BY option are automatically defined.

Integer-1 and integer-2 may be signed. Integer-1 must be positive. All identifiers must be either index data items or numeric elementary items described as integers; however, identifier-4 must not name an index data item.

Format 1 Considerations

When the SET statement is executed, one of the following actions occurs:

- Index-name-1 is converted to a value that corresponds to the same table element to which either index-name-3, identifier-3, or integer-1 corresponds. If identifier-3 is an index data item, no conversion takes place.
- If identifier-1 is an index data item, it is set equal to either the contents of index-name-3 or identifier-3 when identifier-3 is also an index data item. Integer-1 cannot be used in this case.
- If identifier-1 is not an index data item, it is set to an occurrence number that corresponds to the value of index-name-3. Neither identifier-3 nor integer-1 can be used in this case.

Format 2 Considerations

When the SET statement is executed, the contents of index-name-4 are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of integer-2 or identifier-4. The value of the index must correspond to an occurrence number of an element in the associated table.

IBM Extension:

Format 3 Considerations

Each mnemonic-name must be associated with an external switch (UPSI-0 through UPSI-7), the status of which can be altered.

The status of each external switch is modified to an 'ON' status if the ON phrase is specified, or an 'OFF' status if the OFF phrase is specified.

Format 4 Considerations

Each condition-name must be associated with a conditional variable.

The literal in the VALUE clause associated with condition-name is moved to the conditional variable in accordance with the rules for elementary moves. If more than one literal is specified in the VALUE clause, the first literal in that VALUE clause is moved.

SORT/MERGE

Arranging records in a particular order or sequence is a common requirement in data processing; such record ordering can be accomplished using sorting or merging operations. While both operations accomplish record ordering, the functions and capabilities of a sort and a merge are different.

A sort produces an ordered file from one to eight input files that may be completely unordered as to sort sequence. Thus, the sort operation must accept unordered sort input and produce ordered sort output.

A merge produces an ordered file from two to eight input files, each of which is already ordered in the merge sequence.

> **IBM Extension:** Input files do not need to be sequenced prior to a merge operation.

COBOL has special language features that assist in sort and merge operations so that the user need not program these operations in detail. The System/34 Sort Utility must be installed on the system when COBOL sort/merge functions are executed.

For an explanation of messages that are issued by the System/34 Sort Utility after it has been invoked by a COBOL object program, see the *Sort Reference Manual* or the *Displayed Messages Guide*.

SORT/MERGE CONCEPTS

Sorting and merging have always constituted a large percentage of the workload in business data processing. COBOL standardizes the specification of these operations, making them easy to specify and modify. In addition, the COBOL programmer can alternatively use the System/34 Sort Utility to perform these operations as a separate job step. The COBOL language supports these operations through the file-control entry in the Environment Division, the SD (sort-merge-file-description) entry in the Data Division, and the SORT and MERGE statements in the Procedure Division.

The sort or merge file is described through the file-control entry in the Environment Division, and the SD entry in the Data Division. The sort or merge file is the working file used during the sort or merge; it can be considered an internal file. As such, blocking and internal storage allocation for this file are not under the control of the COBOL programmer. However, a sort or merge file, like any file, is a set of records, and a sort-merge file description can be considered a particular type of file description.

The sort-merge file is processed through a Procedure Division SORT or MERGE statement. The statement specifies the key field(s) within the record upon which the sort or merge is to be arranged. Keys can be specified as ascending or descending. When more than one key is specified, a mixture of the two sequences is allowed. The sequence of sorted or merged records conforms to the mixture of keys specified.

Sort Concepts

Through the SORT statement, the COBOL user has access to input procedures (used before sorting) and output procedures (used after sorting) that can add, delete, alter, edit, or otherwise modify the records in the input or output files. A COBOL program can contain any number of sorts, each of them with its own independent input and/or output procedures. During SORT statement execution, these procedures are automatically executed at the specified point in processing; thus, extra passes through the sort file are avoided.

A COBOL program containing a sort is usually organized so that one or more input files are read and operated on by an input procedure. Within the input procedure a RELEASE statement (analogous to the WRITE statement) places a record in the sort file. That is, when input procedure execution is completed, a sort file has been created by placing records one at a time into the sort file through the RELEASE statement. If the user does not wish to modify the records before the sorting operation begins, the SORT statement USING option releases the unmodified records to the sort file.

After all the input records have been placed in the sort file, the sorting operation is executed. This operation arranges the entire set of sort file records in the sequence specified by the key(s).

After completion of the sorting operation, sorted records can be made available from the sort file, one at a time, through a RETURN statement for modification in an output procedure. If the user does not wish to modify the sorted records, the SORT statement GIVING option names the sorted output file.

Note: The Ideographic Sort Utility can be accessed from your COBOL program by the SORT statement. However, the Ideographic Sort Utility will not sort 2-byte characters. It will provide a 1-byte EBCDIC sort.

Merge Concepts

Through the MERGE statement, the COBOL user has access to output procedures (used after merging) that can modify the records in the output file. The COBOL program can contain any number of merge operations, each with its own independent output procedures. During MERGE statement execution, these procedures are automatically executed at the specified point in processing.

MERGE statement execution begins the merge processing. This operation compares keys within the records of the input files and arranges the records within the merged file in the sequence specified by the key(s).

Merged records can be made available, one at a time, through a RETURN statement for modification in an output procedure. If the user does not wish to modify the merged records, the MERGE statement GIVING option names the merged output file.

SORT/MERGE PROGRAMMING CONSIDERATIONS

This section describes considerations for performing sort or merge operations.

Main Storage Requirements

The System/34 Sort Utility is called whenever a COBOL program uses the sort or merge function. Since the Sort Utility requires 14 k bytes, at least this much storage must be allocated to the COBOL program. If the COBOL program uses less than 14 k bytes, a // REGION OCL statement with a size of 14 k bytes or greater should be specified.

Disk Storage Requirements

Whenever an input procedure is used with a SORT statement or an output procedure is used with a SORT or MERGE statement, disk space must be available for COBOL to use for the sort or merge file. A // FILE OCL statement must be present for each such sort or merge file. When both the USING and GIVING options are specified for a SORT or MERGE statement, the system automatically allocates an intermediate work area to hold the sort or merge records. Thus, a // FILE OCL statement is not required for such a sort or merge file.

The System/34 Sort Utility called by the COBOL program requires the following disk work areas:

- An area to hold the COBOL program. The Sort Utility overlays the COBOL program. Thus, the COBOL program must be saved on disk before processing begins. To do this, the Sort Utility allocates a scratch file large enough to hold the COBOL program. To determine the size of the scratch file in blocks, the number of k bytes in the program is multiplied by 0.4. If the product is not a whole number, it is rounded up to the next integer.

- A work area in which to perform the sort. This can be created explicitly by specifying a // FILE OCL statement with a NAME-WORK parameter. If this statement is not used, the Sort Utility allocates a scratch file large enough to perform the desired sort or merge operation. See the *Sort Reference Manual* for details on the size of this work file.

Note: A // RESERVE OCL statement, if used, can reserve disk space for the scratch files that the Sort Utility uses. If not enough space is reserved, an error message is issued. This message lets the user allocate more space or cancel the job.

Performance Considerations

Improved performance can generally be obtained by specifying the USING or GIVING option on the SORT or MERGE statement. The best performance can generally be obtained by using both options. This lets the programmer bypass writing the input records into the sort or merge work area (via RELEASE statements) and reading the sorted records from the sort or merge work area (via RETURN statements).

When using input and output procedures, a BLOCK CONTAINS clause cannot be specified on the SD statement for the sort or merge file. System/34 COBOL always defaults to one record, the minimum blocking factor. When many records will be processed in an input or output procedure, a different manner of coding might better control the blocking.

For example, if file 1 is a very large file that is to be reformatted, sorted, and written into file 2, a normal technique would be to write the SORT statement with an input procedure, GIVING file 2. The input procedure would read file 1, reformat the data, and release the records to the sort file. The Sort Utility would then sort the data in the sort file and write this data into file 2.

Because of the minimum blocking factor on the sort or merge file, it is possible that a faster-running program might be obtained by reading file 1, reformatting the data, and writing the data into file 2 before coding the SORT statement. The SORT statement could then be coded USING file 2 and GIVING file 2. In this case, performance can be improved by either of the following:

- Increasing the value of the BLOCK CONTAINS clause in your program

- Specifying a BLOCK CONTAINS clause with a value greater than one if your program does not already contain this clause

ENVIRONMENT DIVISION–SORT/MERGE

In the Environment Division, the programmer must write file-control entries for each file used as input to or output from a sort or merge operation. The programmer must also write a file-control entry for each unique sort-file or merge-file.

File-Control Paragraph

For a description of input and output files of a sort or merge operation, see the *FILE-CONTROL Paragraph* in Chapter 3.

I-O-Control Paragraph

In the I-O-Control Paragraph, the SAME SORT AREA or SAME SORT-MERGE AREA clause is used.

Format

The SAME SORT AREA or SAME SORT-MERGE AREA clause functions to reduce storage area assignment to a given SORT or MERGE statement.

In the SAME AREA clause, SORT and SORT-MERGE are equivalent.

The SAME SORT AREA or SAME SORT-MERGE AREA clause specifies one storage area available for sort/merge operations by each named sort or merge file. That is, the storage allocated for one such operation is available for reuse in another.

When the SAME SORT AREA or SAME SORT-MERGE AREA clause is specified, at least one file-name specified must name a sort or merge file. Files that are not sort or merge files can also be specified. The following rules apply:

- More than one SAME SORT AREA or SAME SORT-MERGE AREA clause can be specified; however, one sort or merge file must not be named in more than one such clause.
- If a file that is not a sort or merge file is named in both a SAME AREA clause and in one or more SAME SORT AREA or SAME SORT-MERGE AREA clauses, all the files in the SAME AREA clause must also appear in that SAME SORT AREA or SAME SORT-MERGE AREA clause.
- Files named in a SAME SORT AREA or SAME SORT-MERGE AREA clause need not have the same organization or access.
- Files named in a SAME SORT AREA or SORT-MERGE AREA clause that are not sort or merge files do not share storage with each other unless the user names them in a SAME AREA or SAME RECORD AREA clause.
- Files named in a SAME SORT AREA or SAME SORT-MERGE AREA clause that are not sort or merge files must not be open during the execution of a SORT or MERGE statement that refers to a sort or merge file named in the clause.

Rules for the specification of SAME RECORD AREA clause are given under *I-O-Control Paragraph* in Chapter 3.

This page is intentionally left blank.

DATA DIVISION–SORT/MERGE

In the File Section, the programmer must write an FD entry for each file that is input to or output from the sort/merge operation, as well as a record description entry. In addition, there must be an SD (sort-merge-file-description) entry for each sort or merge file.

Format

```
[SD file-name
    [RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS]
    [DATA {RECORD IS / RECORDS ARE} data-name-1 [,data-name-2] . . .] .
{record-description-entry} . . .]
```

The level indicator SD identifies the beginning of the SD entry, and must precede the file-name. The file-name must specify a sort or merge file.

The clauses that follow file-name are optional, and their order of appearance is not significant. Both the RECORD CONTAINS clause and the DATA RECORDS clause are described in Chapter 4.

One or more record description entries must follow the SD entry. However, no input/output statements can be executed for this file.

The following example illustrates the File Section entries needed for a sort or merge file:

```
SD  SORT-FILE.

01  SORT-RECORD  PICTURE X(80).
```

PROCEDURE DIVISION–SORT/MERGE

The Procedure Division contains MERGE and SORT statements to describe the merge and sort operations and, optionally, sort input procedures or sort/merge output procedures. A sort input procedure must contain a RELEASE statement that makes each record available to the sorting operation. A sort/merge output procedure must contain a RETURN statement that makes a sorted/merged record available to the output procedure.

The Procedure Division can contain more than one SORT or MERGE statement. These statements can appear anywhere except in the Declaratives portion or in the sort input or sort/merge output procedures.

Files specified in the USING and GIVING options of the SORT and MERGE statements must be described explicitly or implicitly in their file-control entries as having sequential organization.

USE procedures are not executed if they reference files specified on a USING or GIVING option of a SORT or MERGE statement. The USING or GIVING files are not accessed by COBOL, but by the Sort Utility. If these files are also referenced in an I/O statement within an output procedure or a SORT input procedure, a USE procedure for the file specified is invoked when necessary.

MERGE Statement

The MERGE statement combines from two to eight identically sequenced files that have already been sorted according to an identical set of ascending/descending keys on one or more keys. This statement makes records available in merged order to an output procedure or output file.

Format

```
MERGE file-name-1 ON { ASCENDING  } KEY data-name-1 [ ,data-name-2 ] ...
                     { DESCENDING }

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

[ COLLATING SEQUENCE IS alphabet-name ]

  USING file-name-2, file-name-3 [ ,file-name-4 ] ...

{ OUTPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ] }
{                                     { THRU    }                 }
{ GIVING file-name-5                                              }
```

File-name-1 is the name given in the SD entry that describes the records being merged. No file-name may be repeated in the MERGE statement.

When the MERGE statement is executed, all records contained in file-name-2, file-name-3, and so on, are accepted by the sort/merge program and then merged according to the key(s) specified. These files must not be open when the MERGE statement is executed; they are automatically opened and closed by the MERGE operation, and all implicit functions are performed. The files are closed as if the CLOSE statement is written without any optional processing.

SORT Statement

The SORT statement accepts records from one or more files, sorts them according to the specified key(s), and makes records available either through an output procedure or in an output file. The maximum number of files accepted by the SORT statement is eight.

Format

```
SORT file-name-1 ON { ASCENDING  } KEY data-name-1 [ ,data-name-2 ] . . .
                    { DESCENDING }

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

[COLLATING SEQUENCE IS alphabet-name ]

{ INPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ] }
                                     { THRU    }
{ USING file-name-2 [ ,file-name-3 ] . . .                        }

{ OUTPUT PROCEDURE IS section-name-3 [ { THROUGH } section-name-4 ] }
                                      { THRU    }
{ GIVING file-name-4                                               }
```

File-name-1 is the name given in the SD entry that describes the records being sorted.

When the SORT statement is executed, all records contained in file-name-2, file-name-3, and so on are accepted by the sort/merge program and then sorted according to the key(s) specified. These input files must not be open at the time the SORT statement is executed; they are automatically opened and closed by the SORT operation, and all implicit functions are performed. The files are closed as if the CLOSE statement is written without any optional processing.

MERGE Statement and SORT Statement Options

Most SORT/MERGE statement options apply to both the SORT and the MERGE statements. The common SORT/MERGE statement options are: the ASCENDING/DESCENDING KEY option, the COLLATING SEQUENCE option, the USING option, the GIVING option, and the OUTPUT PROCEDURE option. The INPUT PROCEDURE option applies only to the SORT statements.

ASCENDING/DESCENDING KEY Option

This option specifies that records are to be processed in an ascending or descending sequence based on the specified sort/merge keys.

Each data-name specifies a KEY data item on which the sort-merge will be based. Each such data-name must identify a data item in a record associated with file-name-1. The following rules apply:

- A specific KEY data item must be physically located in the same position and have the same data format in each input file; however, it need not have the same data-name.
- If file-name-1 has more than one record description, then the KEY data items need be described in only one of the record descriptions.
- KEY data items must be fixed-length items.
- KEY data items must not contain an OCCURS clause or be subordinate to an item that contains an OCCURS clause.
- A maximum of 12 KEY data items may be specified.
- The total length of all KEY data items must not exceed 256 bytes. The maximum length of 256 bytes may include bytes used by the compiler. Generally, one additional byte per key is used for each numeric key specified. For a merge operation, 3 additional bytes are used to maintain relative record position. The maximum number of bytes used by the compiler is 15. Therefore, the user has a minimum of 241 bytes, and possibly all 256 bytes, depending on the type of operation and the data types of the keys specified.
- KEY data items may be qualified; they may not be subscripted or indexed.

The KEY data items are listed in order of decreasing significance, regardless of how they are divided into KEY phrases. Using the format as an example, data-name-1 is the most significant key and records are processed in ascending or descending order on that key; data-name-2 is the next most significant key and within data-name-1 records are processed on data-name-2 in ascending or descending order. Within data-name-2, records are processed on data-name-3 in ascending or descending order; within data-name-3, records are processed on data-name-4 in ascending or descending order.

The direction of the sort/merge operation depends on the specification of the ASCENDING or DESCENDING key words as follows:

- When ASCENDING is specified, the sequence is from the lowest key value to the highest key value.
- When DESCENDING is specified, the sequence is from the highest key value to the lowest.
- If the KEY data item is alphabetic, alphanumeric, alphanumeric edited, or numeric edited, the sequence of key values depends on the collating sequence used.
- The key comparisons are performed according to the rules for comparison of operands in a relation condition. See *Relation Condition* in Chapter 5.

COLLATING SEQUENCE Option

This option specifies the collating sequence to be used in nonnumeric comparisons for the KEY data items in this sort/merge operation.

Alphabet-name must be specified in the SPECIAL-NAMES paragraph alphabet-name clause. Any one of the alphabet-name clause options can be specified with the following results.

- When NATIVE is specified, the EBCDIC collating sequence is used for all nonnumeric comparisons.
- When STANDARD-1 is specified, all nonnumeric comparisons are made as if the data items were translated from EBCDIC into ASCII. For more information on the translation of EBCDIC items into ASCII, see Appendix G.
- When the literal option is specified, the collating sequence established by the specification of literals in the alphabet-name clause is used for all nonnumeric comparisons.

When the COLLATING SEQUENCE option is omitted, the PROGRAM COLLATING SEQUENCE clause (if specified) in the OBJECT-COMPUTER paragraph specifies the collating sequence to be used. When both the COLLATING SEQUENCE option and the PROGRAM COLLATING SEQUENCE clause are omitted, the EBCDIC collating sequence is used.

USING Option

When the USING option is specified, all input files are transferred automatically to file-name-1. At the time the SORT or MERGE statement is executed, these input files must not be open; the COBOL compiler opens, reads, and closes these files automatically. If EXCEPTION/ERROR procedures are specified for these files, the COBOL compiler makes the necessary linkage to these procedures.

The input files must have sequential organization.

All input files must be described in an FD entry in the Data Division, and their record descriptions must describe records of the same size as the record described for the sort or merge file. If the elementary items that make up these records are not identical, the user must describe the input records as having an equal number of character positions as the sort record.

GIVING Option

When the GIVING option is specified, all the sorted or merged records in file-name-1 are automatically transferred to the output file (MERGE file-name-5 or SORT file-name-4). At the time the SORT or MERGE statement is executed, this output file must not be open; the COBOL compiler opens, writes, and closes the file automatically. If EXCEPTION/ERROR procedures are specified for the output file, the COBOL compiler makes the necessary linkage to these procedures.

The output file must have sequential organization.

The output file must be described in an FD entry in the Data Division, and its record description(s) must describe records of the same size as the record described for the sort or merge file. If the elementary items that make up these records are not identical, the user must describe the output record as having an equal number of character positions as the sort or merge record.

SORT INPUT PROCEDURE Option

This option specifies the section-name(s) of a procedure that is to modify input records before the sorting operation begins.

Section-name-1 specifies the first (or only) section in the input procedure. Section-name-2 (when specified) identifies the last section of the input procedure.

The input procedure must consist of one or more sections that are written consecutively and do not form a part of any output procedure. The input procedure must include at least one RELEASE statement in order to transfer records to the sort-file.

Control must not be passed to the input procedure except when a related SORT statement is being executed because the RELEASE statement in the input procedure has no meaning unless it is controlled by a SORT statement. The input procedure can include any procedures needed to select, create, or modify records. The following restrictions apply to the procedural statements within an input procedure:

- The input procedure must not contain any SORT or MERGE statements.
- The input procedure must not contain ALTER, GO TO, or PERFORM statements that refer to procedure-names outside the input procedure. The execution of a CALL statement to another program that follows standard linkage conventions, or the execution of USE declaratives is not considered a transfer of control outside an input procedure. Hence, they are allowed to be activated within these procedures.
- The remainder of the Procedure Division must not contain any transfers of control to points inside the input procedure with the exception of the return of control from a Declaratives Section.

If an input procedure is specified, control is passed to the input procedure when the SORT program input phase is ready to receive the first record. The compiler inserts a return mechanism at the end of the last section of the input procedure and when control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted. The RELEASE statement transfers records from the Input Procedure to the sort file, which is then used in the input phase of the sort operation.

SORT/MERGE OUTPUT PROCEDURE Option

This option specifies the section-name(s) of a procedure that is to modify output records from the sort or merge operation.

Section-name-3 specifies the first (or only) section in the output procedure. Section-name-4 (when specified) identifies the last section of the output procedure.

The output procedure must consist of one or more sections that are written consecutively and do not form a part of any input procedure. The output procedure must include at least one RETURN statement in order to make sorted/merged records available for processing.

When all the records are sorted/merged, control is passed to the output procedure. The RETURN statement in the output procedure is a request for the next record.

Control must not be passed to the output procedure except when a related SORT or MERGE statement is being executed because RETURN statements in the output procedure have no meaning unless they are controlled by a SORT or MERGE statement. The output procedure may consist of any procedures needed to select, modify, or copy the records that are being returned one at a time from the sort/merge file. There are three restrictions on the procedural statements within the output procedure:

- The output procedure must not contain any SORT or MERGE statements.
- The output procedure must not contain ALTER, GO TO, or PERFORM statements that refer to procedure-names outside the output procedure. The execution of a CALL statement to another program that follows standard linkage conventions or the execution of USE declaratives is not considered transfers of control outside an output procedure. Hence, they are allowed to be activated within these procedures.
- The remainder of the Procedure Division must not contain any transfers of control to points inside the output procedure with the exception of the return of control from a Declaratives Section.

When an output procedure is specified, control passes to it after the sort/merge file (file-name-1) has been placed in sequence by the sort/merge operation. The COBOL compiler inserts a return mechanism at the end of the last section in the output procedure; when control is passed to the last statement in the output procedure, the return mechanism terminates the sort or merge, and passes control to the next executable statement after the SORT or MERGE statement.

SORT or MERGE INPUT/OUTPUT PROCEDURE Control

The INPUT or OUTPUT PROCEDURE options function in a manner similar to Format 1 of the PERFORM statement (the simple PERFORM). For example, naming a section in an OUTPUT PROCEDURE option causes execution of that section during the sort/merge operation to proceed as if that section were named in a PERFORM statement. As with the PERFORM statement, execution of the section is terminated after execution of its last statement. The last statement in Input and Output Procedures can be the EXIT statement. This is useful for documentation purposes.

RELEASE Statement (Sort Function Only)

The RELEASE statement transfers records from an input/output area to the initial phase of a sort operation.

The RELEASE statement may be specified only within an input procedure associated with a SORT statement. Within an input procedure at least one RELEASE statement must be specified.

When the RELEASE statement is executed, the current contents of record-name are placed in the sort file; that is, made available to the initial phase of the sort operation.

Format

RELEASE record-name [FROM identifier]

Record-name must specify a record associated with the SD entry for file-name-1. Record-name may be qualified.

When the FROM identifier option is specified, the RELEASE statement is equivalent to the statement MOVE identifier to record-name followed by the statement RELEASE record-name. Moving takes place according to the rules for the MOVE statement without the CORRESPONDING option.

Identifier and record-name must not refer to the same storage area.

After the RELEASE statement is executed, the information in record-name is no longer available unless file-name-1 is specified in a SAME RECORD AREA clause, in which case record-name is still available as a record of the other files named in that clause. When the FROM identifier option is specified, the information is still available in identifier.

When control passes from the input procedure, the sort file consists of all those records placed in it by execution of RELEASE statements.

RETURN Statement

The RETURN statement transfers records from the final phase of a sort or merge operation to an input/output area.

The RETURN statement may be specified only within an output procedure associated with a SORT or MERGE statement. Within an output procedure at least one RETURN statement must be specified.

Format

RETURN file-name RECORD [INTO identifier] AT END imperative-statement

When the RETURN statement is executed, the next record from file-name is made available for processing by the output procedure.

File-name must be described in a Data Division SD entry.

If more than one record description is associated with file-name, these records automatically share the same storage; that is, the area is implicitly redefined. After RETURN statement execution, only the contents of the current record are available; if any data items lie beyond the length of the current record, their contents are undefined.

When the INTO identifier option is specified, the RETURN statement is equivalent to the statement RETURN file-name followed by the statement MOVE record-name TO identifier. Moving takes place according to the rules for the MOVE statement without the CORRESPONDING option. Any subscripting or indexing associated with identifier is evaluated after the record has been returned and immediately before it is moved to identifier.

The record areas associated with file-name and identifier must not be the same storage area.

After all records have been returned from file-name, the AT END imperative-statement is executed, and no more RETURN statements may be executed.

Library Copy Facility

Prewritten source program entries can be included in a source program at compile time. Thus, an installation can use standard file descriptions, record descriptions, or procedures without recoding them. These entries and procedures can be saved in user libraries or the system library. They can be included in the source program by means of the COPY statement.

COPY Statement

The COPY statement places previously written text in a COBOL program.

Format

```
COPY text-name [ { OF } library-name ]
               [ { IN }              ]

[ REPLACING { , { ==pseudo-text-1== }  BY { ==pseudo-text-2== } } ... ]
            {   { identifier-1      }     { identifier-2      }
            {   { literal-1         }     { literal-2         }
            {   { word-1            }     { word-2            }
```

Compilation of the source program containing COPY statements is logically equivalent to processing all COPY statements before processing the resulting source program.

The effect of processing a COPY statement is that the library text associated with text-name is copied into the source program, logically replacing the entire COPY statement beginning with the word COPY and ending with the period, inclusive. When the REPLACING option is not specified, the library text is copied unchanged.

The text-name is the name of the source-member to be copied. The text-name must follow the rules for formation of a program-name. The first eight characters of the text-name are used as the identifying name; these first eight characters must, therefore, be unique within one library.

If input from more than one library is desired during compilation, text-name must be qualified by the OF/IN library-name of the library in which it resides. If library-name is not specified, text-name is qualified by the library specified by the LIBRARY option of the PROCESS statement. However, if neither of the above methods are used, the system library is assumed to contain the source member.

The library-name must follow the rules for formation of a program-name. The first eight characters of the library-name are used as the identifying name; these first eight characters must, therefore, be unique within the system. The uniqueness of the text-name and the library-name is determined after the formation and conversion rules for a program-name have been applied. These rules are given under *PROGRAM-ID Paragraph* in Chapter 3.

A COPY statement may appear in the source program anywhere that a character-string or a separator may appear. However, a COPY statement must not be specified within the resulting copied text. Each COPY statement must be preceded by a space, and followed by a period and a space.

Comment lines may appear in library text. Comment lines in library text are copied into the source program unchanged and are interpreted logically as a single space.

Debugging lines may appear in library text. When a COPY statement is specified on a debugging line, the copied text is treated as though it appeared on a debugging line except that comment lines in the library text appear as comment lines in the resulting source program.

The syntactic correctness of the entire COBOL source program cannot be determined until all COPY statements have been completely processed, because the syntactic correctness of the library text cannot be independently determined.

Library text copied from the library is placed into the same area of the resultant program as it is in the library. Library text must conform to the rules for standard COBOL format.

REPLACING Option

In the REPLACING option, each operand may consist of one of the following: pseudo-text, an identifier, a literal, or a COBOL word. When the REPLACING option is specified, each operand-1 from the library text is replaced by its associated operand-2.

Pseudo-text is a sequence of character-strings and/or separators bounded by, but not including, pseudo-text delimiters (==). Both characters of each pseudo-text delimiter must appear on one line; however, character-strings within pseudo-text can be continued.

IBM Extension: Division, section, and paragraph entries, when comprising pseudo-text-1, may follow format rules; these entries, when comprising pseudo-text-2, must follow format rules. That is, each word in pseudo-text-2 that is copied into the program is placed in the same area of the resultant program as it appears in pseudo-text-2. Example 2 illustrates this concept.

Pseudo-text-1, which is the library text, must be one or more words; that is, it must not be null; neither can it consist solely of the space character and/or of comment lines.

Pseudo-text-2 may be zero, one, or more words; that is, it may consist of nulls, space characters, or comment lines.

Each identifier may be defined in any Data Division section.

Each literal may be numeric or nonnumeric.

Each COBOL word may be any single COBOL word.

Programming Notes

Sequences of code (such as file and data descriptions, error and exception routines, and so on) that are common to a number of programs can be cataloged and used in conjunction with the COPY statement. If naming conventions are established for such common code, then the REPLACING option need not be specified. If the names will change from one program to another, then the REPLACING option can be used to supply meaningful names for this program.

REPLACING Option Processing

When the REPLACING option is specified, the library text is copied, and each properly matched occurrence of operand-1 within the library text is replaced by the associated operand-2.

For purposes of matching, each identifier-1, literal-1, or word-1 is treated as pseudo-text containing only identifier-1, literal-1, or word-1 respectively. Separator spaces in identifiers are optional in both the library text and the comparison text.

The comparison proceeds as follows:

- Any separator comma, semicolon, and/or space preceding the leftmost word in the library text is copied into the source program. Beginning with the leftmost library text word and the first operand-1 specified in the REPLACING option, the entire REPLACING operand that precedes the key word BY is compared to an equivalent number of contiguous library text words.

- Operand-1 matches the library text only if the ordered sequence of text words in operand-1 is equal, character for character, to the ordered sequence of library words. For matching purposes, each occurrence of a comma or semicolon separator is considered to be a single space. However, when operand-1 consists solely of a separator comma or semicolon, it participates in the match as a text word. In this case, the space following the comma or semicolon separator can be omitted. Each sequence of one or more space separators is considered to be a single space.

- If no match occurs, the comparison is repeated with each successive operand-1 (if specified) until either a match is found or there are no further REPLACING operands.

- Whenever a match occurs between operand-1 and the library text, the associated operand-2 is copied into the source program in the place of operand-1.

  IBM Extension: Operand-2 is copied in the place of operand-1 unless psuedo-text-2 positioning rules cause the replacement to be inserted in a different area.

- When all operands have been compared and no match is found, the leftmost library text word is copied into the source program.

- The next successive uncopied library text word is then considered the leftmost text word, and the comparison process is repeated, beginning with the first operand-1. The process continues until the rightmost library text word has been compared.

- A comment line occurring in operand-1 and in the library text is interpreted for matching purposes as a single space. A comment line appearing in operand-2 is copied unchanged into the source program.

- Debugging lines are not permitted in operand-1. Debugging lines, however, are permitted in library text and in operand-2. Text words in a debugging line are matched as if no D appeared in column 7.

- Text words after replacement are placed in the source program according to standard COBOL format rules.

Notes:

1. Arithmetic and logical operators are considered to be text words and can be replaced only through the pseudo-text option.
2. When a figurative constant is operand-1, it will match only exactly as specified. For example, if ALL 'AB' is specified in the library text, then 'ABAB' is not considered a match. Only ALL 'AB' is considered a match.
3. When replacing a PICTURE character-string, the pseudo-text option must be used. To avoid ambiguities, pseudo-text-1 must specify the entire PICTURE clause, including the key word PICTURE or PIC.
4. When replacing a comment-entry in the Identification Division, the pseudo-text option must be used.

COPY Statement Example

Example 1: In this example, the library entry PAYLIB consists of the following Data Division entries:

```
02 B PIC S99.
02 C PIC S9(5)V99.
02 D PIC S9999 OCCURS 1 TO 52 TIMES
     DEPENDING ON B OF A.
```

The programmer can use the COPY statement in the Data Division of a program as follows:

```
01 PAYROLL. COPY PAYLIB.
```

In this program, the library entry is then copied. The resulting entry is treated as if it had been written as follows:

```
01 PAYROLL.
   02 B PIC S99.
   02 C PIC S9(5)V99.
   02 D PIC S9999 OCCURS 1 TO 52 TIMES
        DEPENDING ON B OF A.
```

To change some (or all) of the names within the library entry, the programmer can use the REPLACING option:

```
01 PAYROLL. COPY PAYLIB REPLACING A BY
   PAYROLL B BY PAY-CODE C BY GROSS-PAY.
```

In this program, the library entry is copied. The resulting entry is treated as if it had been written as follows:

```
01 PAYROLL.
   02 PAY-CODE    PIC S99.
   02 GROSS-PAY   PIC S9(5)V99.
   02 D           PIC S9999 OCCURS 1 TO 52
                  TIMES DEPENDING ON
                  PAY-CODE OF PAYROLL.
```

The changes shown are made only for this program. The entry as it appears in the library remains unchanged.

Example 2: Library-member A consists of the following COBOL text:

```
Area    Area
 A       B
 |       |
 v       v
          02   ABC PIC 99.99.
```

A COPY statement that replaces Area-B level 02 with Area-A level 01 would be written as follows:

```
Area    Area
 A       B
 |       |
 v       v
          COPY A REPLACING == 02 == BY ==
01 ==
```

The resulting source would be

```
Area    Area
 A       B
 |       |
 v       v
01          ABC PIC 99.99
```

Segmentation Feature

The Segmentation Feature provides programmer-controlled storage optimization of the Procedure Division by allowing that division to be subdivided, for overlays both physically and logically.

Although segmentation is not required, the Procedure Division of a source program is usually written as a consecutive group of sections. Each section is composed of a series of closely related operations that perform a particular function.

When the Segmentation Feature is used, the entire Procedure Division must be divided into sections. Each section in the division must be classified according to physical and logical attributes by a system of segment numbers. Segment numbers must be in the range 0 through 99.

Program Segments

Two types of program segments are available: permanent and independent.

Permanent Segments (0-49)

A permanent segment composes the fixed portion (sometimes called root segment) of the object program. A permanent segment cannot be overlaid by any other part of the program. It is always present in its last-used state.

The fixed portion is the part of the object portion that logically resides in main storage during execution of the object program.

Independent Segments (50-99)

An independent segment is defined as the part of the object program that can overlay or be overlaid by another independent segment. An independent segment is always considered to be in its initial state each time it is made available to the program.

Segmentation Logic

In a segmented program, the sections are classified by a system of segment numbers. All sections with the same segment-number constitute a program segment with that priority. The segment-number must be an integer ranging in value from 0 through 99. Segments with segment-numbers ranging from 0 through 49 are called permanent segments. Segments with segment-numbers ranging from 50 through 99 are independent segments. If a segment-number is omitted from the section header, the segment-number is assumed to be zero. Sections in the declaratives portion of the Procedure Division must contain segment-numbers less than 50.

The following criteria should be used in assigning segment numbers and segment types:

- Frequency of Use–Sections that are used often or sections that must be available for references at all times, should be within permanent segments. Less frequently used sections should be within independent segments.
- Frequency of Reference–The more frequently a section is referred to, the lower its segment number; the less frequently the section is referred to, the higher its segment number.
- Logical relationships–Sections that frequently communicate with each other should be given identical segment numbers (sections with the same segment-number do not need to be adjacent in the source program).

Segmentation Control

The logical sequence of the program is the same as the physical sequence of the program except for specific transfers of control. A reordering of the object module is necessary if a given segment has its section scattered throughout the source program. When the object module is thus reordered, the compiler provides control transfers to maintain the logic flow of the source program. The compiler inserts instructions necessary to load and/or initialize a segment when necessary. Within a source program, control may be transferred to any paragraph in a section. It is not necessary to transfer control to the beginning of a section.

Execution of the segmented object program begins in the fixed portion. All tables, literals, and data buffers are included in the fixed portion. Called object-time subroutines are also part of the root segment. When CALL statements appear in a segmented program, subprograms are loaded with the fixed portion of the main program.

If a segmented program calls a subprogram, the CALL statement may appear in any segment. However, the overlay linkage editor forces all such subprograms into the fixed portion.

When a segmented COBOL subprogram contains independent segments, the following link-edit technique must be used:

1. Both called and calling programs must be compiled with the OBJECT option in the PROCESS statement to create a relocatable module (subroutine member).

2. The Overlay Linkage Editor is invoked using OCL or the OLINK command.

3. The OCL MODULE statement specifies the mainline module name and the names of any subprogram modules that contain independent segments.

Refer to the *IBM System/34 Overlay Linkage Editor* for additional information.

Executable Object Program Size

When using the Segmentation Feature, the user can direct the overlay linkage editor to construct an object program of a particular size. The user does this by specifying the size of main storage available for execution in the MEMORY SIZE clause of the OBJECT-COMPUTER paragraph. The overlay linkage editor then attempts to produce a program that will utilize the available space, making resident all the sections that it can based on segment numbers. The user is informed if the program does not fit in the space specified. If the MEMORY SIZE clause is omitted, the size of the compiler region is assumed.

PROCEDURE DIVISION–SEGMENTATION

In the Procedure Division of a segmented program, segments are classified by segment-numbers. The segment-number is included in the section header.

Format

section-name SECTION [segment-number] .

Special Considerations–Segmentation

When segmentation is used, there are restrictions on the ALTER, PERFORM, and SORT and MERGE statements. Transfer of control, calling programs, and called programs also require special consideration.

ALTER Statement

A GO TO statement within an independent segment may be changed only by an ALTER statement that is within the same segment. A GO TO statement in a permanent segment may be changed by an ALTER statement in any segment of the program.

An altered GO TO statement is always returned to its original state during program execution except when the PERFORM statement is in a permanent segment and the altered GO TO statement is in an independent segment. The GO TO statement retains its altered state until the conclusion of the PERFORM statement.

PERFORM Statement

A PERFORM statement in a permanent segment can only refer to sections wholly contained within the fixed portion of the program or sections wholly contained within one independent segment.

A PERFORM statement in an independent segment can only refer to sections wholly contained within the fixed portion of the program or sections wholly contained within the same independent segment as the PERFORM statement.

For each execution of a PERFORM statement, control is passed to the performed procedures only once.

Note: When a PERFORM statement references procedures within an independent segment, control must not pass outside that independent segment. Also, when a PERFORM statement in an independent segment references the fixed portion, control must not pass to another independent segment from the fixed portion. Return linkages may be destroyed, and a processor check is likely to occur.

SORT and MERGE Statements

If a SORT or MERGE statement appears in the fixed portion, then any SORT input procedures or SORT/MERGE output procedures must appear completely in either the fixed portion or one independent segment.

If a SORT or MERGE statement appears in an independent segment, then any SORT input procedures or SORT/MERGE output procedures must appear completely in either the fixed portion or the same independent segment as the SORT or MERGE statement.

Transfer of Control

The Segmentation Feature imposes no restrictions on transfers of control as long as the control path remains within the range of permanent segments. Permanent segments are logically identical and can be treated by the user as though the program were not segmented.

If independent segments are involved in the transfer of control, restrictions do apply. Independent segments are loaded into main storage under control of the system management routines. The user must assume that logically only one independent segment at a time is present in main storage. This may not be the case physically. The user must not execute statements that depend on the presence of any given independent segment (other than the one in which the statements appear) in main storage at any given time.

Calling and Called Programs

The CALL statement may appear anywhere within a segmented program. When a CALL statement appears in an independent segment, that segment is in its last-used state when control is returned to the calling program.

A called program may not be segmented.

Inter-program Communication

Complex data processing problems are often solved by the use of separately compiled but logically interdependent programs which, at execution time, form logical and physical subdivisions of a single run unit. A run unit is the total machine-language program necessary to solve a data processing problem; it includes one or more object programs, and may include object programs from source programs written in System/34 FORTRAN IV and System/34 Basic Assembler.

SUBPROGRAM LINKAGE CONCEPTS

When the solution of a problem is subdivided into more than one program, the constituent programs must be able to communicate with each other through transfers of control and/or through reference to common data.

Transfers of Control

In the Procedure Division, a calling program can transfer control to a called program, and a called program may itself transfer control to yet another called program. However, a called program must not directly or indirectly call its caller. For example, if program A calls program B; program B calls program C; and program C then calls program A the results are unpredictable.

When control is passed to a called program, execution proceeds in the normal way. When a called program processing is completed, the program can either transfer control back to the calling program, call another program, or end the run unit.

Common Data

Program interaction may require that both programs have access to the same data.

In a calling program, the common data items are described in the same manner as other File and Working-Storage Section items. Storage is allocated for these items in the calling program.

In a called program, common data items are described in the Linkage Section. Storage is not allocated to them in the called program. Because a calling program may itself be a called program, common data items may be described in the Linkage Section of the calling program. In this case, storage is not allocated for these items in the calling program itself, but rather in the program that called the calling program. For example, program A calls program B which calls program C. Data items in program A can be described in the Linkage Sections of programs B and C, and the one set of data can be made available to all three programs.

When control is transferred from the calling to the called program, the programmer must furnish a list of the common data items in both programs. The sequence of identifiers in both lists determines the match of identifiers between the calling and called programs. A corresponding pair of identifiers in the list names a single set of data that is available to both programs. While the called program is executing, any reference to one of these identifiers is a reference to the corresponding data of the calling program.

COBOL Language Considerations

In the Data Division of the source programs, the programmer defines the common data items to be used by both the calling and called programs. In the calling program, these items can be defined in the File, Working-Storage, or Linkage Sections. In the called program, these items must be defined in the Linkage Section. Common data items need not have the same name and data description, but they must contain the same number of characters.

In the Procedure Division, the list of common data items is established through the USING option, which names those data items available to both programs. In the called program, only those items named in the USING list of the called program are available from the data storage of the calling program. Figure 6-4 illustrates this concept.

A CALL statement in the calling program transfers control to the first nondeclarative procedural statement in the called program. When the called program has completed execution, control is returned to the calling program by an EXIT PROGRAM statement. The entire run unit can be ended by a STOP RUN statement in either program.

Calling Program Description	Called Program Description
WORKING-STORAGE SECTION. 01 PARAM-LIST. 05 PARTCODE PIC A. 05 PARTNO PIC X(4). 05 U-SALES PIC 9(5). . . . PROCEDURE DIVISION. . . . CALL 'CALLPG' USING PARAM-LIST.	LINKAGE SECTION. 01 USING-LIST. 10 PART-ID PIC X(5). 10 SALES PIC 9(5). . . . PROCEDURE DIVISION USING USING-LIST.
Note: In the calling program, the code for parts (PARTCODE) and the part number (PARTNO) are referred to separately. In the called program, the code for parts and the part number are combined into one data item (PART-ID); therefore in the called program, a reference to PART-ID is the only valid reference to them.	

Figure 6-4. Common Data Items in Subprogram Linkage

System Considerations

The main COBOL program and all called programs are part of the same load module. When control is transferred to the called program, it is already resident in storage, and a branch to the called program takes place. Subsequent executions of the CALL statement make the called program available in its last-used state.

Note: Through the use of the OLINK command, it is possible to write nonresident called programs into the same area in main storage. For more information on this process, see *Link-Editing with Overlay* in Chapter 9.

DATA DIVISION–SUBPROGRAM LINKAGE

In the Data Division of a called program, the programmer specifies in the Linkage Section those data items that are common with the calling program.

Format

```
LINKAGE SECTION.

    [data-item-description-entry ] . . .

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

The Linkage Section has meaning only if this object program functions under control of a CALL statement that contains the USING option.

The Linkage Section describes data available within the calling program and referred to in both the calling and called programs. Items described in the Linkage Section do not have space allocated for them in the called program. Procedure Division references to these data items are resolved at object time by equating the reference in the called program to the location used in the calling program. For index-names, no such correspondence is established. Index-name references in the calling and called programs always refer to separate indexes.

Items defined in the Linkage Section can be referred to in the Procedure Division only if they are one of the following:

- Operands of a USING option in this program
- Data items subordinate to such a USING option operand
- Items associated with such a USING operand (such as condition-names or index-names)

Each Linkage Section record-name and noncontiguous data-name must be unique, because neither can be qualified. Descriptions of each clause valid in the Linkage Section are given under *Data Description* in Chapter 4. The following additional considerations apply.

Record Description Entries

Items that have a hierarchical relationship with one another must be grouped into level-01 records according to the rules for formation of record descriptions. Data description clauses can be used to complete the description of the entry. Except for level-88 condition-names, the VALUE clause must not be specified.

Data Item Description Entries

Items that have no hierarchical relationship with each other can be defined as noncontiguous items with level-number 77. The following clauses are required:

- Level-number 77
- Data-name
- PICTURE or USAGE IS INDEX

Other data description clauses are optional and, when necessary, can complete the description of the item. Except for level-88 condition-names, the VALUE clause must not be specified.

PROCEDURE DIVISION–SUBPROGRAM LINKAGE

In the Procedure Division, control is transferred between COBOL object programs by means of the CALL statement.

Reference to common data is provided through the USING option, which can be specified in the CALL statement and in the Procedure Division header of the called program.

The EXIT PROGRAM statement allows termination of called program processing. The STOP RUN statement allows termination of the run unit.

CALL Statement

The CALL statement causes control to be transferred from one object program to another within the run unit. The calling program must contain a CALL statement at the point where another program is to be called.

Execution of the CALL statement causes control to pass to the first nondeclarative instruction of the called program. Control returns to the calling program at the instruction following the CALL statement.

Called programs themselves can contain CALL statements, but a called program must not contain a CALL statement that directly or indirectly calls the calling program.

A given calling program can call up to 20 subprograms.

Format

```
CALL literal-1 [ USING data-name-1 [ ,data-name-2 ] . . . ]
```

Literal-1 must be nonnumeric and must conform to the rules for formation of a program-name. The first six characters of the literal are used to make the correspondence between the calling program and the called program. The literal must specify the program-name of the called subprogram.

CALL statement execution causes control to pass to the called subprogram. The first time a called program is entered, its state is that of a fresh copy of the program. Each subsequent time a called program is entered, the state is as it was upon the last exit from that program. Thus, the reinitialization of the following items is the responsibility of the programmer:

- GO TO statements that have been altered
- Data items
- PERFORM statements

USING Option

The USING option makes data items from a calling program available to the called program. If the called program has no need of data items from the calling program, the USING option can be omitted. If data must be passed, the USING option must appear in two places:

- The CALL statement of the calling program
- The Procedure Division header of the called program.

The identifiers specified in the USING option of the Procedure Division header must be data items defined in the Linkage Section of the called program. The identifiers specified in the calling program can be defined in the File, Working-Storage, or Linkage Section.

Identifiers must be level 01 or 77 items. They can be qualified.

IBM Extension: The data-names in the USING option of the CALL statement in the calling program can have level-numbers other than 01 or 77. These data-names can be indexed or subscripted.

The data-names specified in the USING option of the Procedure Division header must be data items defined in the Linkage Section of the called program with a level-number 01 or 77.

The number of identifiers in the USING option of the CALL statement must equal the number of data-names in the USING option of the Procedure Division header. The maximum number of identifiers, or data-names that can be specified is 15. If the number of identifiers does not equal the number of data-names, unpredictable results may occur. Also, the data descriptions of the identifiers and data-names must correspond by position.

The names of identifiers and data-names need not correspond, but the same identifier or data-name must not appear more than once in the same USING option.

EXIT PROGRAM Statement

The EXIT PROGRAM statement specifies the logical end of a called program.

Format

```
paragraph-name. EXIT PROGRAM.
```

The EXIT statement must be preceded by a paragraph-name, and it must be the only statement in the paragraph.

If control reaches an EXIT PROGRAM statement while operating under the control of a CALL statement, control returns to the point in the calling program immediately following the CALL statement. If control reaches an EXIT PROGRAM statement and no CALL statement is active, control passes through the exit point to the first sentence of the next paragraph.

STOP RUN Statement

The STOP RUN statement is discussed under the *STOP Statement* in Chapter 5.

Segmentation Considerations

A CALL statement may appear anywhere within a segmented program; the compiler ensures that the proper logic flow is maintained. Therefore, if a CALL statement appears in an independent segment, that segment is made available in its last-used state when control is returned from the called program.

SUBPROGRAM LINKAGE FEATURE EXAMPLES

The CALL statement is illustrated in the following program example.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CALLSTAT.
    .
    .
    .
DATA DIVISION.
    .
    .
    .
WORKING-STORAGE SECTION.
01  RECORD-2 PIC X.
01  RECORD-1.
    05 SALARY PICTURE S9(5)V99.
    05 RATE    PICTURE S9V99.
    05 HOURS  PICTURE S99V9.
    .
    .
    .
PROCEDURE DIVISION.
    .
    .
    .
    CALL 'SUBPRG' USING RECORD-1, RECORD-2.
    .
    .
    .
STOP RUN.
```

The following called subprogram is associated with the preceding calling program.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUBPRG.
    .
    .
    .
DATA DIVISION
    .
    .
    .
LINKAGE SECTION.
01  PAYREC.
    10 PAY              PICTURE S9(5)V99.
    10 HOURLY-RATE     PICTURE S9V99.
    10 HOURS           PICTURE S99V9.
77  CODE PIC X.
    .
    .
    .
PROCEDURE DIVISION USING PAYREC, CODE.
    .
    .
    .
    EXIT PROGRAM.
    .
    .
    .
```

Processing begins in the calling program, CALLSTAT. When the first CALL statement is executed, control is transferred to the first statement of the Procedure Division in SUBPRG, which is the called program.

When SUBPRG receives control, the values within RECORD-1 are made available to SUBPRG; however, in SUBPRG they are referred to as PAYREC. The data-items within PAYREC and CODE contain the same number of characters as RECORD-1 and RECORD-2, although the descriptions are not identical. When processing within SUBPRG reaches the EXIT PROGRAM statement, control is returned to the calling program.

Debugging Features

The Debugging Features specify the conditions under which procedures are to be monitored during program execution.

COBOL source language debugging statements are provided. The user decides what to monitor and what information to retrieve for debugging purposes. The COBOL debugging features simply provide access to pertinent information.

COBOL SOURCE LANGUAGE DEBUGGING

COBOL language elements that implement the Debugging Feature are a compile-time switch (WITH DEBUGGING MODE), an object-time switch, a USE FOR DEBUGGING Declarative, the special register DEBUG-ITEM, and debugging lines that can be written in the Environment, Data, and Procedure Divisions.

Compile-Time Switch

In the SOURCE-COMPUTER paragraph of the Configuration Section, the WITH DEBUGGING MODE clause acts as a compile-time switch.

Format

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

The WITH DEBUGGING MODE clause serves as a compile-time switch for the debugging statements written in the source program.

When WITH DEBUGGING MODE is specified, all debugging sections and debugging lines are compiled as specified in this chapter. When WITH DEBUGGING MODE is omitted, all debugging sections and debugging lines are treated as documentation.

Object-Time Switch

When execution of the object program begins, a prompt that asks whether the debugging statements are to be activated or bypassed is issued to the requesting operator (or system operator for a batch job or an MRT program). The response acts as an object-time switch.

When debugging mode is specified, through the object-time switch, all the debugging sections and debugging lines compiled into the object program are activated.

When debugging mode is suppressed, through the object-time switch, any USE FOR DEBUGGING declarative procedures are inhibited. However, all debugging lines remain in effect.

Recompilation of the source program is not required to activate or deactivate the object-time switch.

When WITH DEBUGGING MODE is not specified in the SOURCE-COMPUTER paragraph, the object-time switch is not available.

USE FOR DEBUGGING Declarative

The USE FOR DEBUGGING sentence in the Procedure Division identifies the items in the source program that are to be monitored by the associated debugging Declarative procedure.

Format

```
USE FOR DEBUGGING ON { procedure-name-1   }
                     { ALL PROCEDURES     }

    [ , procedure name 2 ] . . . .
```

When specified, all debugging sections must be written immediately after the DECLARATIVES header. Except for the USE FOR DEBUGGING sentence there must be no reference to any nondeclarative procedure within the debugging procedure.

Automatic execution of a debugging section is not caused by a statement appearing in a debugging section.

A debugging section is not executed more than once as the result of the execution of a single statement. For a PERFORM statement that causes repeated execution of a procedure, any associated procedure-name debugging declarative section is executed once for each repetition.

For debugging purposes, each separate occurrence of an imperative verb within an imperative statement begins a separate statement.

Statements appearing outside the debugging sections must not refer to procedure-names defined within the debugging sections.

Except for the USE FOR DEBUGGING sentence itself, statements within a debugging declarative section may refer to procedure-names defined in a different USE procedure only through the PERFORM statement. Procedure-names within debugging declarative sections must not appear in any USE FOR DEBUGGING sentence.

Figure 6-5 defines the points during program execution when the USE FOR DEBUGGING procedures are executed. The procedure-name-n refers to the first and all subsequent specifications of that type of operand in one use for DEBUGGING sentence.

USE FOR DEBUGGING operand	Upon execution of the following, the USE FOR DEBUGGING procedures are executed immediately:
procedure-name-n	Before each execution of the named procedure. After execution of every ALTER statement except ALTER statements in Declarative procedures.
ALL PROCEDURES	Before each execution of every nondebugging procedure. After execution of every ALTER statement except ALTER statements in Declarative procedures.

Figure 6-5. Execution of Debugging Declaratives

When ALL PROCEDURES is specified in a USE FOR DEBUGGING sentence, procedure-name-1, procedure-name-2, and so on, must not be specified in any USE FOR DEBUGGING sentence. The ALL PROCEDURES option may be specified only once in a program.

References to the DEBUG-ITEM special register may only be made from within a debugging declarative procedure.

Any procedure-name may appear in only one USE FOR DEBUGGING sentence, and it may appear only once in that sentence.

When a USE FOR DEBUGGING operand is used as a qualifier, such reference in the program does not activate the debugging procedures.

DEBUG-ITEM Special Register

The DEBUG-ITEM special register provides information for a debugging declarative procedure. DEBUG-ITEM has the following implicit description.

```
01  DEBUG-ITEM.
    02  DEBUG-LINE          PICTURE IS X(6).
    02  FILLER              PICTURE IS X VALUE SPACE.
    02  DEBUG-NAME          PICTURE IS X(30).
    02  FILLER              PICTURE IS X VALUE SPACE.
    02  DEBUG-SUB-1         PICTURE IS S9999 SIGN IS
                             LEADING SEPARATE CHARACTER.
    02  FILLER              PICTURE IS X VALUE SPACE.
    02  DEBUG-SUB-2         PICTURE IS S9999 SIGN IS
                             LEADING SEPARATE CHARACTER.
    02  FILLER              PICTURE IS X VALUE SPACE.
    02  DEBUG-SUB-3         PICTURE IS S9999 SIGN IS
                             LEADING SEPARATE CHARACTER.
    02  FILLER              PICTURE IS X VALUE SPACE.
    02  DEBUG-CONTENTS      PICTURE IS X(n).
```

The DEBUG-ITEM special register provides information about the conditions causing debugging section execution.

Before each debugging section is executed, DEBUG-ITEM is filled with spaces. The contents of the DEBUG-ITEM subfields are then updated according to the rules for the MOVE statement, with one exception: DEBUG-CONTENTS is updated as if the move were an alphanumeric to alphanumeric elementary move without conversion of data from one form of internal representation to another. After updating, each field contains:

- DEBUG-LINE: The compiler-generated sequence number.
- DEBUG-NAME: The first 30 characters of the name causing debugging section execution. All qualifiers are separated by the word OF.
- DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3: These fields contain spaces.
- DEBUG-CONTENTS: Data is moved into DEBUG-CONTENTS as shown in Figure 6-6.

Item Causing Debug Section Execution	DEBUG-LINE Contains Number of COBOL Statement Referring to	DEBUG-NAME Contains	DEBUG-CONTENTS Contains
procedure-name-n ALTER reference	ALTER statement	procedure-name-n	procedure-name-n in TO PROCEED TO phrase
GO TO procedure-name-n	GO TO statement	procedure-name-n	blanks
procedure-name-n in SORT/MERGE INPUT/OUTPUT PROCEDURE	SORT/MERGE statement	procedure-name-n	'SORT INPUT' 'SORT OUTPUT' 'MERGE OUTPUT' as applicable
PERFORM statement transfer of control	this PERFORM statement	procedure-name-n	'PERFORM LOOP'
procedure-name-n in a USE procedure	statement causing USE procedure execution	procedure-name-n	'USE PROCEDURE'
implicit transfer from previous sequential procedure	previous statement executed in previous sequential procedure (see note)	procedure-name-n	'FALL THROUGH'
1st execution of 1st nondeclarative procedure	line number of first statement in the procedure	first nondeclarative procedure-name	'START PROGRAM'
Note: If this paragraph is preceded by a section header and control is passed through the section header, the statement number refers to the section header.			

Figure 6-6. DEBUG-ITEM Subfield Contents

A simple way to use DEBUG-ITEM for debugging is demonstrated in Figure 6-7.

```
SEQUENCE        CONT.  A   B                COBOL STATEMENT
(PAGE) SERIAL
1  3  4  6      7      8   12  16  20  24  28  32  36  40  44  48  52  56  60
   0  1          PROCEDURE DIVISION.
   0  2
   0  3          DECLARATIVES.
   0  4              DEBUGG SECTION. USE FOR DEBUGGING ON ALL PROCEDURES.
   0  5                  DISPLAY DEBUG-ITEM.
   0  6
```

Figure 6-7. Using DEBUG-ITEM to Trace Program Flow Conditioned by Object-time DEBUG Mode

Debugging Lines

A debugging line is any line in a source program with a D coded in column 7 (the continuation area). If a debugging line contains nothing but spaces in Area A and Area B, it is considered a blank line.

Each debugging line must be written so that a syntactically correct program results whether the debugging lines are compiled into the program or treated as documentation.

Successive debugging lines are permitted. Debugging lines may be continued. However, each continuation line must contain a D in column 7, and character-strings must not be broken across two lines.

Debugging lines may be specified only after the OBJECT-COMPUTER paragraph.

When the WITH DEBUGGING MODE clause is specified in the SOURCE-COMPUTER paragraph, all debugging lines are compiled as part of the object program.

When the WITH DEBUGGING MODE clause is omitted, all debugging lines are treated as documentation.

IBM Extension: Three debug language statements are added to COBOL to assist in the debugging of a program. The statements are READY TRACE, RESET TRACE, and EXHIBIT. These statements can be used as often as necessary and are normally interspersed throughout a COBOL source program. The output produced by the TRACE and EXHIBIT statements is output to the current SYSLIST device.

TRACE Statement

The formats of the READY TRACE statement and the RESET TRACE statement are as follows:

Format

```
READY TRACE.

RESET TRACE.
```

The READY TRACE statement causes the compiler-generated internal statement number for each section-name and paragraph-name to be displayed. These statement numbers are listed at execution time when control passes to these sections and paragraphs. Hence, the output of the READY TRACE statement appears as a list of statement numbers.

To reduce the length of the list and the time taken to generate it, a trace can be stopped with a RESET TRACE statement. The READY TRACE-RESET TRACE combination is helpful in examining a particular area of the program where the flow of control is difficult to determine (for example, where code consists of a series of PERFORM statements or nested conditional statements). The READY TRACE statement can be coded so that the trace begins before control passes to that area. The RESET TRACE statement can be coded so that the trace stops when the program has passed beyond that area.

If a TRACE statement appears anywhere in the source program, a push-down list of the last paragraphs and sections executed is maintained at execution time. The list is printed when a paragraph or section is entered and READY TRACE has just gone into effect.

To ensure that this list is maintained during debugging, the programmer should insert a TRACE statement at some point in the Procedure Division. For example, a RESET TRACE statement can always be included at the beginning of the Procedure Division.

After debugging, the user should remove all TRACE statements from the program to save both storage and time in the actual production program.

The following is an example of the output of the TRACE statement.

```
 STNO  A...B... COBOL SOURCE STATEMENTS...

 30    PROCEDURE DIVISION.
 31    SECTION-1 SECTION.
 32    PARA-1.
 33        DISPLAY 'PARA-1 ENTERED'.
 34        READY TRACE.
 35    PARA-2.
 36        DISPLAY 'PARA-2 ENTERED'.
 37    PARA-3.
 38        DISPLAY 'PARA-3 ENTERED'.
 39        RESET TRACE.
 40    PARA-4.
 41        DISPLAY 'PARA-4 ENTERED'.
 42        READY TRACE.
 43    PARA-5.
 44        DISPLAY 'PARA-5 ENTERED'.
 45        STOP RUN.
```

TRACE Output

```
PARA-1 ENTERED
**STNO=00035, PRECEDING WERE 00032
   00031
PARA-2 ENTERED
**STNO=00037
PARA-3 ENTERED
PARA-4 ENTERED
**STNO=00043, PRECEDING WERE 00040
      00037  000035  00032  00031
PARA-5 ENTERED
```

EXHIBIT Statement

A user can display upon the current SYSLIST device the value of a data item during program execution by using the EXHIBIT statement.

Format

```
EXHIBIT { NAMED                } identifier-1 [ identifier-2] ...
        { CHANGED NAMED        }
```

- EXHIBIT NAMED option displays the names and values of the identifiers listed in the statement.
- EXHIBIT CHANGED NAMED option displays the names and values of the identifiers listed in the statement only if the value has changed since the last execution of the statement. The comparison is based on the leading 256 character positions of the identifier. When such a statement is first executed, all values are considered changed and are displayed.

Data values can be used to check the accuracy of the program. For example, using EXHIBIT NAMED, the user can display specified fields from records, compute the calculations himself, and compare his calculations with the output from his program. The coding for a payroll problem might be:

```
.
.
DATA DIVISION.

01  PAYRCDHRS.
    05 EMP-NO PIC 999 USAGE IS COMP.
    .
    .
    05 RATE-PER-HOUR....
    05 HRSWKD....
    05 OVERTIMERHRS....
    05 GROSS-PAY....
.
.
PROCEDURE DIVISION.
.
.
GROSS-PAY-CALC.
    COMPUTE GROSS-PAY = RATE-PER-HOUR *
       (HRSWKD + 1.5 * OVERTIMEHRS).
    PERFORM TEST1.
NET-PAY-CALC.
    .
    .
TEST1   IF (EMP-NO +9) / 10 IS NOT EQUAL TO
        EMP-NO / 10 EXHIBIT NAMED
        RATE-PER-HOUR, HRSWKD,
        OVERTIMEHRS, GROSS-PAY ELSE NEXT
        SENTENCE.
EXIT.
```

This coding causes the values of the four fields to be listed for the first and for every tenth data record before net pay calculations are made. The output could appear as:

```
RATE-PER-HOUR = 4.00
HRSWKD = 40.0
OVERTIMEHRS = 0.0
GROSS-PAY = 160.00

RATE-PER-HOUR = 4.10
HRSWKD = 40.0
OVERTIMEHRS = 1.5
GROSS-PAY = 173.23

RATE-PER-HOUR = 3.35
HRSWKD = 40.0
OVERTIMEHRS = 0.0
GROSS-PAY = 134.00
```

Note: Decimal points are included in this example for clarity, but actual printouts depend on the data descriptions of the fields in the Data Division.

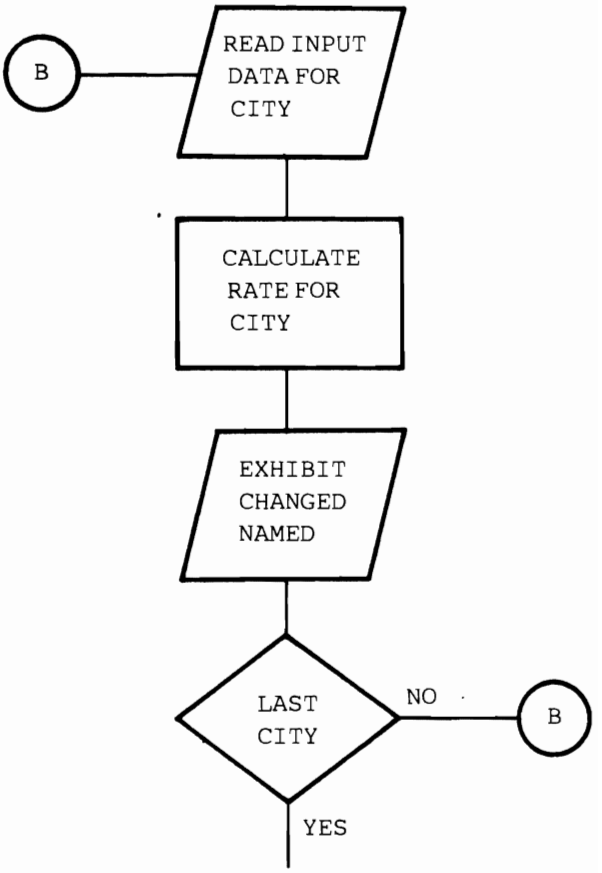
The preceding was an example of checking at regular intervals (every tenth record). A check of any unusual conditions can be made by using various combinations of COBOL statements. For example:

```
IF OVERTIMEHRS IS GREATER THAN 2
  EXHIBIT NAMED PAYRCDHRS.....
```

In connection with the previous example, this statement could cause the entire pay record to be displayed whenever an unusual condition (overtime exceeding two hours) is encountered.

The EXHIBIT statement with the CHANGED NAMED option can also be used to monitor conditions that do not occur at regular intervals. The names and values of identifiers are listed only if the value has changed since the last execution of the statement.

For example, suppose the program calculates postage rates to various cities. The flow of the program might be:



The EXHIBIT statement with the CHANGED NAMED option in the program might be:

```
EXHIBIT CHANGED NAMED STATE CITY RATE
```

The output from the EXHIBIT statement with the CHANGED NAMED option would appear as shown in the following figure. The value of an identifier is listed only if it has changed since the previous execution. For example, since the postage rate to city 02 and city 03 in state 01 are the same, the rate is not printed for city 03.

```
STATE = 01
CITY = 01
RATE = 10
CITY = 02
RATE = 15
CITY = 03
CITY = 04
RATE = 10
STATE = 02
CITY = 01
CITY = 02
RATE = 20
CITY = 03
RATE = 15
CITY = 04
STATE = 03
CITY = 01
RATE = 10
.
.
.
```

FIPS Flagger

The FIPS Flagger, depending on the compiler option chosen, identifies source statements and clauses that do not conform to the federal standard, FIPSPUB 21-1.

1975 FIPS COBOL (Federal Information Processing Standard COBOL), December 1975–is a compatible subset of American National Standard COBOL, X3.23-1974. 1975 FIPS COBOL is subdivided into four levels: full, high-intermediate, low-intermediate, and low. Any program written to conform to 1975 FIPS COBOL must conform to one of these levels of 1975 FIPS COBOL processing. Figure 6-7 shows the 1974 Standard COBOL processing modules included in each of the levels of 1975 FIPS COBOL.

1974 Standard Module	Full FIPS Module	High Intermediate FIPS Module	Low Intermediate FIPS Module	Low FIPS Module
2 NUC 1,2 (Nucleus)	2 NUC 1,2	2 NUC 1,2	1 NUC 1,2	1 NUC 1,2
2 TBL 1,2 (Table Handling)	2 TBL 1,2	2 TBL 1,2	1 TBL 1,2	1 TBL 1,2
2 SEQ 1,2 (Sequential I-O)	2 SEQ 1,2	2 SEQ 1,2	1 SEQ 1,2	1 SEQ 1,2
2 REL 0,2 (Relative I-O)	2 REL 0,2	2 REL 0,2	1 REL 0,2	–
2 INX 0,2 (Indexed I-O)	2 INX 0,2	–	–	–
2 SRT 0,2 (Sort-Merge)	2 SRT 0,2	1 SRT 0,2	–	–
1 RPW 0,1 (Report Writer)	–	–	–	–
2 SEG 0,2 (Segmentation)	2 SEG 0,2	1 SEG 0,2	1 SEG 0,2	–
2 LIB 0,2 (Library)	2 LIB 0,2	1 LIB 0,2	1 LIB 0,2	–
2 DEB 0,2 (Debug)	2 DEB 0,2	2 DEB 0,2	1 DEB 0,2	–
2 IPC 0,2 (Inter-program Communication)	2 IPC 0,2	2 IPC 0,2	1 IPC 0,2	–
2 COM 0,2 (Communications)	2 COM 0,2	2 COM 0,2	–	–

Figure 6-7. The 1974 Standard and 1975 FIPS COBOL

A compiler option can be specified that causes elements that exceed a specified level of 1975 FIPS COBOL to be flagged. The following elements are flagged as exceeding the COBOL level they are listed under.

1975 High FIPS COBOL Flagging

When flagging for the high FIPS level is specified, the following IBM Extensions, if present, are identified.

Global Items

Apostrophe used as quote

Identification Division

Nothing is flagged in the Identification Division.

Environment Division

SYSTEM-CONSOLE, REQUESTOR, CSP, C01, LOCAL-DATA, ATTRIBUTE-DATA, SYSTEM-SHUTDOWN, or UPSI-0 through UPSI-7 in SPECIAL-NAMES
ORGANIZATION IS TRANSACTION
FILE STATUS with second data-name
CONTROL-AREA clause in FILE-CONTROL entry (TRANSACTION files)

Data Division

Unequal level-numbers at same group level
Use of VALUE clause as comment in other than condition-name entries
Asterisk as zero suppression symbol and BLANK WHEN ZERO clause in same entry
Numeric literal without sign in VALUE clause
COMP-3 option of the USAGE clause
COMP-4 option of the USAGE clause
COMPUTATIONAL-3 option of the USAGE clause
COMPUTATIONAL-4 option of the USAGE clause
PICTURE character 1 (Boolean data)
INDICATOR clause
FILLER used as group item

Procedure Division

THEN as separator in an IF statement
FOR clause of ACCEPT or DISPLAY
EXHIBIT statement
TRACE statement
Any reference to a TRANSACTION file (ACQUIRE, CLOSE, DROP, OPEN, READ, USE, WRITE)
SET mnemonic-name TO ON or OFF
SET condition-name TO TRUE
Indexed or subscripted data-name in USING option of CALL statement
Data-name with level-number other than 01 or 77 in USING option of CALL statement

1975 High-Intermediate FIPS COBOL Flagging

When flagging for the high-intermediate FIPS level is specified, all elements in the preceding list are flagged, plus the following FIPS high level COBOL source elements:

Global Items

REPLACING option of COPY statement
OF or IN option of COPY statement

Identification Division

Nothing is flagged in the Identification Division.

Environment Division

SORT or SORT-MERGE option of SAME clause
RECORD KEY clause
INDEXED ORGANIZATION clause in SELECT sentence
RECORD option of SAME clause (indexed or sort files)
ACCESS MODE IS DYNAMIC (indexed files)
RESERVE clause in SELECT sentence (indexed files)

Data Division

Integer-1 TO option of BLOCK CONTAINS clause (indexed files)

Procedure Division

Noncontiguous segments
MERGE statement
KEY option of READ statement
Non-declarative portion of program may contain only SORT and STOP RUN statements if SORT statement is used
Use of more than one SORT statement
COLLATING SEQUENCE in SORT statement
SORT using file-name series
Any reference to an indexed file (CLOSE, DELETE, OPEN, READ, REWRITE, START, USE, or WRITE)

1975 Low-Intermediate FIPS COBOL Flagging

When flagging for the low-intermediate FIPS level is specified, all elements in the preceding lists are flagged, plus the following FIPS high-intermediate level COBOL source elements:

Global Items

Comma or semicolon as punctuation
Continuation of words or numeric literals
Figurative constant ALL literal
Figurative constant HIGH-VALUES
Figurative constant LOW-VALUES
Figurative constant QUOTES
Figurative constant SPACES
Figurative constant ZEROES
Figurative constant ZEROS

Identification Division

DATE-COMPILED paragraph

Environment Division

RESERVE clause in SELECT sentence (sequential or relative files)
OPTIONAL in SELECT sentence
RECORD option of SAME clause (sequential or relative files)
Literal phrase in alphabet-name clause
ACCESS MODE IS DYNAMIC (relative files)
MULTIPLE FILE clause

Data Division

SD level indicator
One-digit level-number
Level-number greater than 10
Data-name beginning with nonalphabetic character
66 or 88 special level-number
Data-name option of VALUE OF clause
ASCENDING or DESCENDING KEY option of OCCURS clause
DEPENDING ON option of OCCURS clause
LINAGE clause
Nesting of REDEFINES clauses
Value clause with THRU option
RENAMES clause
Integer-1 TO option of BLOCK CONTAINS clause (sequential or relative files)

Procedure Division

Qualification of data-names and paragraph-names
CORRESPONDING option
Unary operators
Use of AND OR and NOT in conditional relation
Condition-name condition
Use of arithmetic and relational symbols
(+, -, *, **, /, >, <, =)
Sign condition
ON OVERFLOW statement
FROM in ACCEPT statement
DAY DATE or TIME in ACCEPT statement
Multiple results in ADD statement
GIVING series in ADD statement
Multiple operands in ALTER statement
COMPUTE statement
UPON option of DISPLAY statement
REMAINDER in DIVIDE statement
INTO or GIVING series of DIVIDE statement
GO TO statement with no object
IF statement nesting
Series in INSPECT statement
BY or GIVING series in MULTIPLY statement
EXTEND option of OPEN statement
UNTIL option of PERFORM statement
VARYING option of PERFORM statement
NEXT option of READ statement (relative files)
RELEASE statement
RETURN statement
SEARCH statement
SORT statement
START statement (relative files)
STRING statement
Multiple results in SUBTRACT statement
GIVING series in SUBTRACT statement
UNSTRING statement
EXTEND option of USE statement
EOP or END-OF-PAGE option of WRITE statement
File-name series in USE statement (relative or sequential files)
Comparison of operands of unequal size
Multiple results in MULTIPLY statement
File-name series in CLOSE statement (sequential files)
LOCK phrase in CLOSE statement (sequential files)
File-name series in OPEN statement (sequential files)
Identifier or mnemonic-name as WRITE statement phrase

1975 Low FIPS COBOL Flagging

When flagging for the low FIPS level is specified, all elements in the preceding lists are flagged, plus the following FIPS low-intermediate level COBOL source elements:

Global Items

Debug items
COPY statement
D in continuation area (debugging lines)

Identification Division

Nothing is flagged in the Identification Division

Environment Division

RANDOM option of ACCESS MODE IS clause
WITH DEBUGGING MODE clause
RELATIVE ORGANIZATION clause in SELECT sentence
RELATIVE KEY clause

Data Division

Linkage Section
Reference to relative files

Procedure Division

USING phrase on Procedure Division header
Segment number on section header
CALL statement

EXIT PROGRAM statement

INVALID KEY option of READ statement
INVALID KEY option of REWRITE statement
USE FOR DEBUGGING statement
DEBUG-ITEM special registers
Any reference to a relative file (OPEN, CLOSE, USE, DELETE, READ, WRITE, or REWRITE)

Chapter 7. Transaction File Considerations and Sample Program

The TRANSACTION file is an IBM extension that allows you to read data from and write data to a display station. You define the constants and fields that appear on the display screen with display screen format specifications. You can either enter the display screen format specifications explicitly or generate them through the Screen Design Aid. (For more information on SDA, see the *Screen Design Aid Programmer's Guide and Reference Manual.*) These display screen format specifications are compiled by the $SFGR utility of the system support program (for more information on display screen format specifications and the $SFGR utility, see the *System Support Reference Manual*).

The TRANSACTION file also allows your program to pass data to and read data from another application program through the use of the Interactive Communications Feature (SSP-ICF). Your program can communicate with a program running on the same Sytem/34 or with a program running on another system. In this chapter, the word device means both display stations and SSP-ICF sessions. For a further description and examples of the Interactive Communications Feature, see the *Interactive Communications Feature Reference Manual.*

Note: A TRANSACTION file program includes the main program and all its subprograms.

SUMMARY OF MAJOR LANGUAGE EXTENSIONS

The System/34 COBOL Program Product includes language extensions that support display stations and the Interactive Communications Feature without the use of the System/34 Work Station Support Subroutines (PRPQ). The major extensions are:

- File definition using the file control entries SELECT and ASSIGN.
- A new file organization called TRANSACTION. This file organization allows the user to define one file that supports one or more display stations, one or more SSP-ICF sessions, or any combination of display stations and SSP-ICF sessions.
- Extended file status support for TRANSACTION file processing.
- The ability to access ATTRIBUTE-DATA and the display station local data areas through two new mnemonics that are used with the low-volume input/output verbs, ACCEPT and DISPLAY.
- Standard error handling with the USE procedure in the DECLARATIVES Section and FILE STATUS.
- Extensions to the READ and WRITE verbs perform additional functions associated with TRANSACTION file support of display stations and SSP-ICF sessions.
- The ability to acquire and release devices through two new verbs, ACQUIRE and DROP.
- Support of SFGR indicators via a new data type, Boolean data.

PROGRAM ATTRIBUTES

COBOL programs that use a TRANSACTION file can have one or several requestors. A requestor is the device that initiates, or requests, a program. A program that allows only one requestor is a single requestor terminal program (an SRT). A program that allows more than one requestor is a multiple requestor terminal program (an MRT).

The SRT and MRT attributes are assigned to a program on the COBOL command statement or the COMPILE OCL statement. The number specified for the MRTMAX parameter of either statement is the maximum number of requestors that can be attached to the program at one time. If the MRTMAX parameter value is missing or equal to zero, the program is an SRT.

Note: To change an SRT program to an MRT program, the program must be recompiled (using either the COBOL command statement or the COMPILE OCL statement) with a MRTMAX value greater than or equal to one. In addition, the procedure used to execute the program must be specified as an MRT when you sign off of SEU.

SRT (SINGLE REQUESTOR TERMINAL) PROGRAM

An SRT program is specified if the COBOL command statement or the COMPILE OCL statement does not include an MRTMAX parameter when the program is compiled or if an MRTMAX value of 0 is specified. Although an SRT program allows only one requestor, more than one device can execute the program at the same time. If a second device requests the program, the system support program loads and initiates a second copy of the program. A new copy of the program is loaded for each additional requestor.

An SRT program can have multiple devices attached to it during execution. In this situation, the requestor calls the program, and other devices are acquired for the program through the use of the WORKSTN OCL statement or the COBOL ACQUIRE statement. A display station can be acquired if it is not attached to an executing program and if it is in standby mode. An SSP-ICF session can be acquired if it is not attached to a requestor. Any device, whether acquired or a requestor, can be released from the program through the use of the DROP statement.

When coding SRT programs, the following should be considered:

- If the program is called by more than one requestor, each requestor executes a separate copy of the program.
- The requestor provides the display station local data area.
- Program error messages go to the requestor of the program.
- The program, including any acquired devices, is suspended via inquiry.

MRT (MULTIPLE REQUESTOR TERMINAL) PROGRAM

An MRT program is specified by the MRTMAX parameter on the COBOL command statement or the COMPILE OCL statement. The value specified for the MRTMAX parameter limits the number of requesting devices that the program can process concurrently. An MRTMAX value of one is valid and means that only one device can request the program. In this case, multiple copies of the program are not initiated when additional devices use the same procedure to call the MRT program.

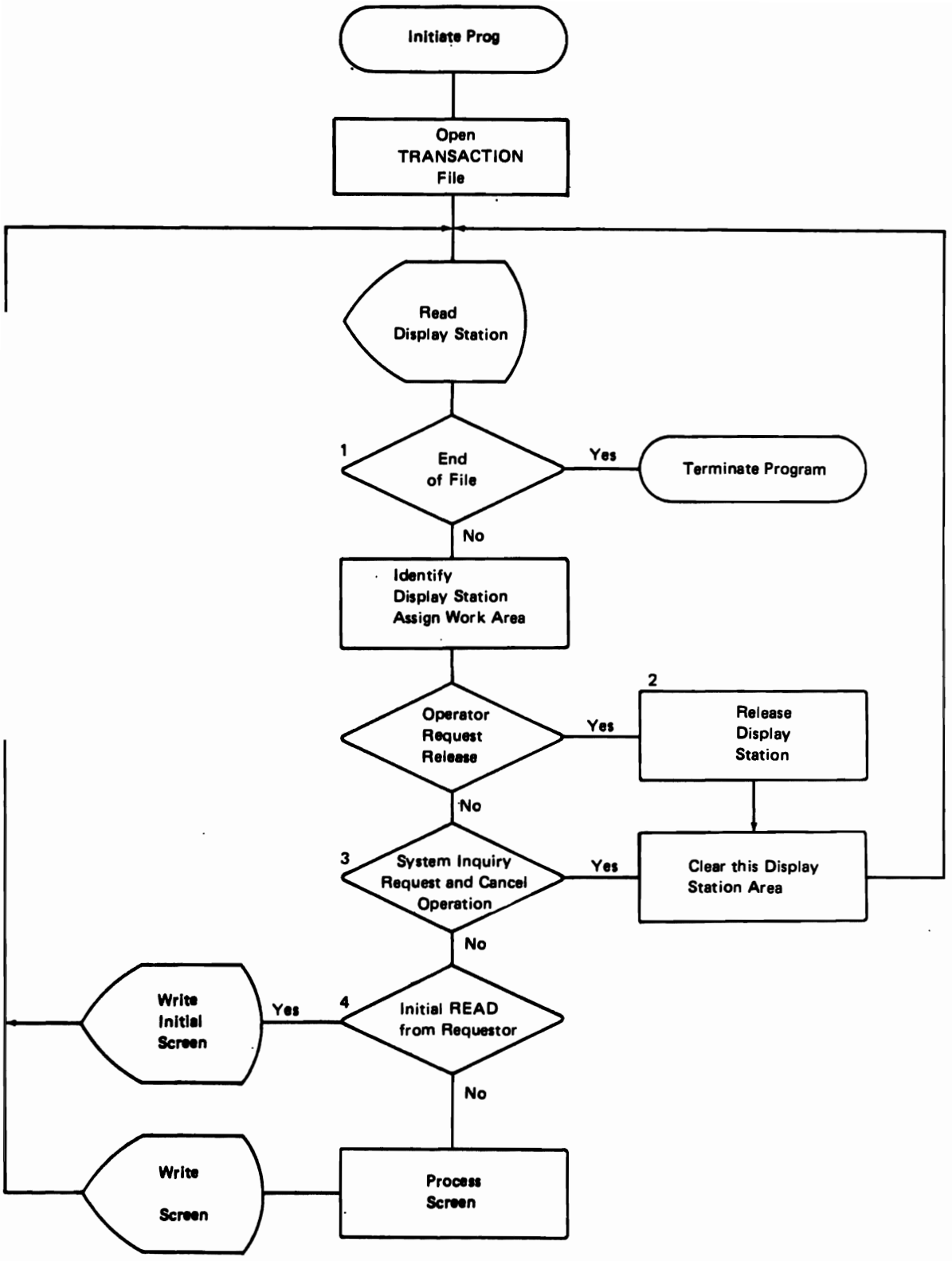
Note: Ordinarily, different MRT procedures should not be used to call the same MRT program because more than one copy of the program could be in storage at the same time.

Each requestor of an MRT program is attached to the same copy of the program during execution. The first requestor of an MRT program causes the program to be loaded and initiated. Each succeeding requestor is attached to the executing program when a read operation is performed. If the program is handling the maximum number of requestors, the system support program queues the additional requestors. When the program releases a requestor, the program processes the next requestor waiting on the queue. For an example of MRT program logic, see Figure 7-1.

When coding MRT programs, the following should be considered:

- If the program is called by more than one requestor, the first requestor causes the program to be initiated.
- A COBOL MRT program must provide the data areas and logic for managing multiple requestors. For an example of code that supports multiple requestors, see lines 56, 59, 104, and 105 in Figure 7-13.
- After the first requestor, each succeeding requestor is attached to the program when a read operation is performed.
- There should be only one READ operation in an MRT program.
- Program error messages go to the system console operator.
- Requestors can leave the program without suspending the program or other devices.
- When inquiry is used in an MRT program, the processing of information from the device requesting the interrupt is suspended. The program continues to process information from other program-requesting devices.

Note: If a display station is released from the program, input cannot be read from and output cannot be written to the display station. If you attempt to do so, a return code is written into the STATUS area or an execution-time error message is displayed.



1–Signaled by AT END condition on TRANSACTION file read
2–Released by DROP operation
3–Test for value of 9D in FILE STATUS
4–Test for value of 01 in FILE STATUS

Figure 7-1. Sample MRT Logic

ATTACHING A DEVICE TO A PROGRAM

Any device can be attached to a COBOL program in one of two ways:

- The device operator calls or requests the program.
- The COBOL program attempts to acquire a specified device when the ACQUIRE statement is used in the Procedure Division. A display station can be acquired if it is not attached to another program and if it is in standby mode. An SSP-ICF session can be acquired if it is not attached to an executing program and if it was specified on the SESSION OCL statement for the job step. The ATTRIBUTE-DATA can be tested to see if a device can be acquired.

A display station can be attached to a COBOL program in one of two additional ways:

- The procedure that is called to execute the COBOL program includes a WORKSTN OCL statement with the parameter REQD-YES. The display station is attached to the program by SSP (system support program product) and must be available (in standby mode) for the program to be initiated.
- The procedure that is called to execute the COBOL program includes a WORKSTN OCL statement with the parameter REQD-NO. REQD-NO indicates that the COBOL program, rather than SSP, attempts to acquire the display station. If the acquire attempt by the COBOL program fails, the display station is deleted from the file.

If a TRANSACTION file program is initiated from the input job queue, a display station must be attached to the program by a WORKSTN OCL statement or by the ACQUIRE statement. If a display station is not attached by one of these methods, undesirable results can occur.

WRITING A PROGRAM WITH A TRANSACTION FILE

Creating a Display Screen Format

The first step in writing a COBOL program that uses a TRANSACTION file is to design the display screen formats that the program uses. You must decide where on the display screen you want the constants and fields for each format to appear. You can use the display screen layout sheet (GX21-9174) as shown in Figure 7-2 to help layout the fields.

Once the formats are designed, you must define them to the System/34 in one of two ways. You can either fill out the two parts (S specifications and D specifications) of the display screen format specifications (see Figure 7-3), or you can use the Screen Design Aid (SDA) to interactively design, define, and compile the display screen formats.

SDA allows you to define the appearance and attributes of a display screen format before the format is actually generated. SDA generates the display screen format specifications automatically, based on how you designed your screen formats. You do not have to fill out the display screen format specification sheets. (For a complete description of SDA, see the *Screen Design Aid Programmer's Guide and Reference Manual.*)

Display Screen Layout Sheet

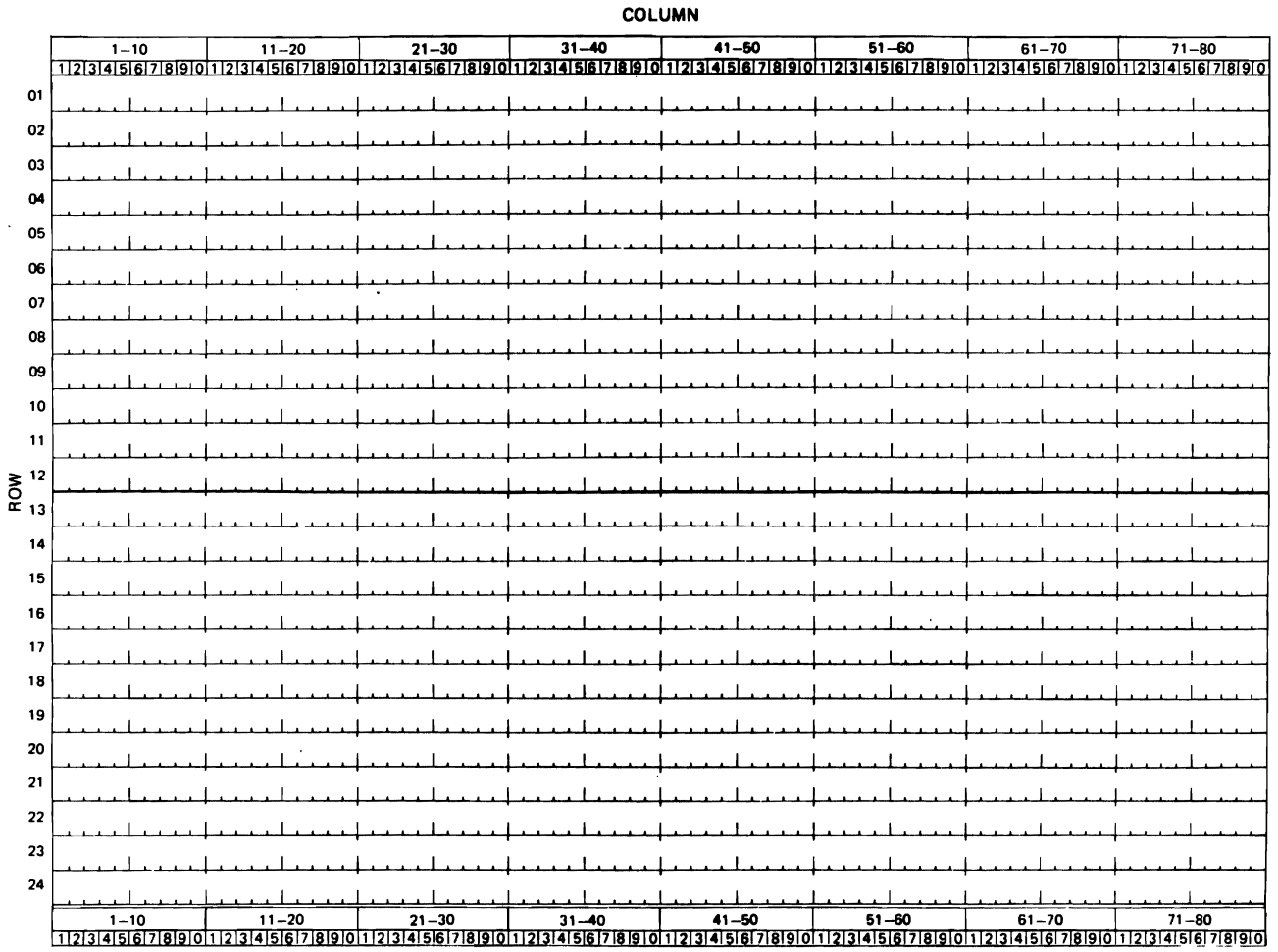
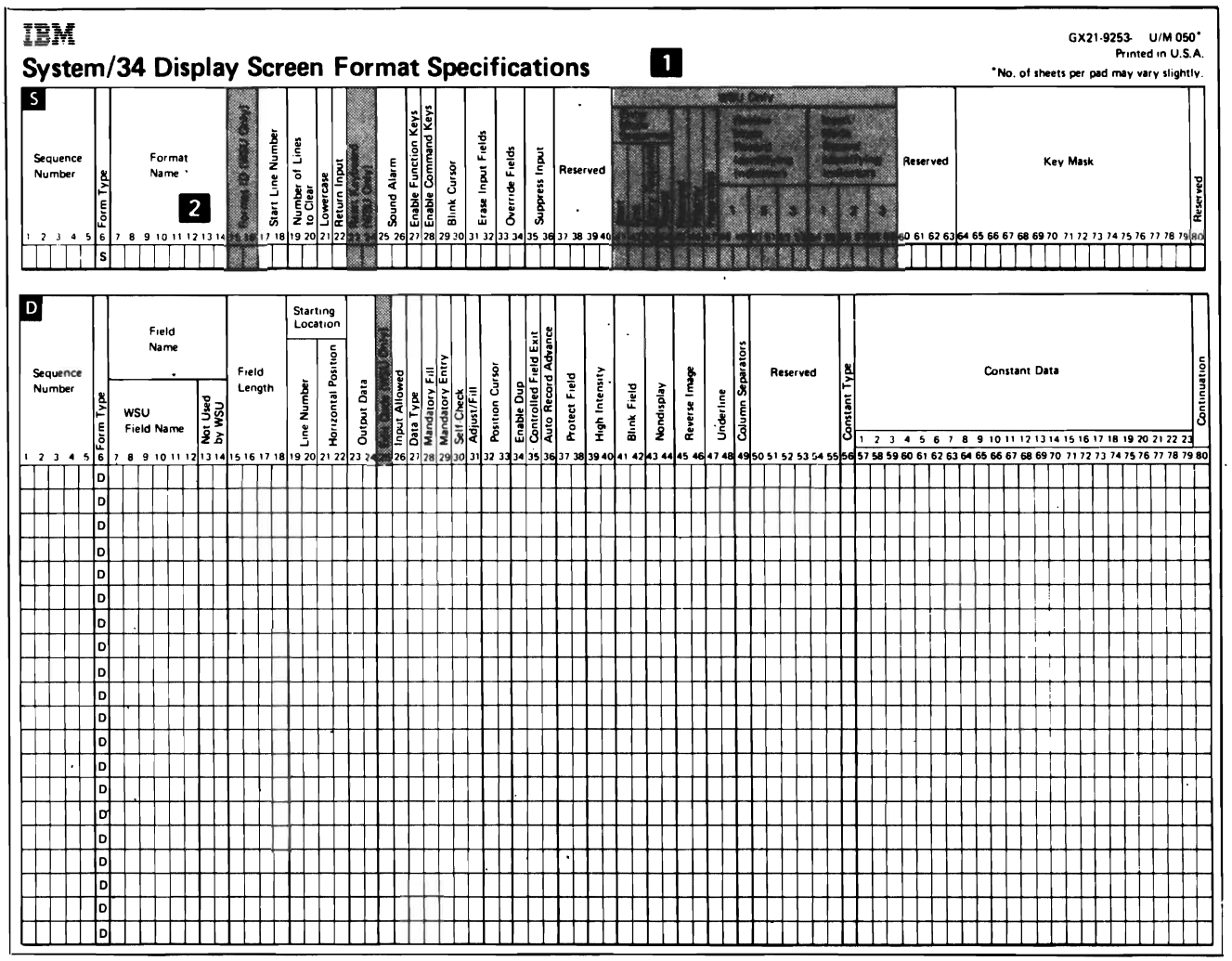


Figure 7-2. Display Screen Layout Sheet



Notes:

1. The shaded columns are not used by $SFGR to generate the screen formats to be used by a COBOL program. This field is not required if the file is to be used with SSP-ICF sessions only.
2. The display screen format name used in columns 7 through 14 of the S specification must be entered on the FORMAT option of the WRITE statement.

Figure 7-3. Display Screen Format Specifications

The COBOL program treats all the information passed to it from one screen format as a single record. Each field in this record must be described on the display screen format specifications (either coded or generated). The D specifications describe the fields and their attributes. You can specify three kinds of fields:

- An *output field* contains information that cannot be changed by the operator. For a normal output operation, the data in the field is supplied either as part of the field definition when the format is generated (a constant) or is supplied by the COBOL program (variable information). If the data is supplied by the COBOL program, it must be moved into a record area before it is written to the display screen. This record area can then be written to the screen at the same time the format is written. The data must be sent to the screen in the same order that it is coded on the D specification for that screen; that is, the order of the output fields in the output record must be the same as that coded on the SFGR D specification.

- An *input field* is a field on the display screen reserved for keyboard entry. Data is entered by the display station operator. The field must be described in the Data Division. If more than one input field is specified for a screen, the input fields should be described in the Data Division in the same order as they are specified in the D specifications for the screen format. If the input fields are not in the order specified in the D specifications, the data could be grouped differently in the TRANSACTION file record.

- An *output/input* field is both displayed (output) on the display screen and read (input) back into the program. The display station operator can key new data over the data that is displayed by the program. This data is then read back into the program.

  Note: Exercise caution when referring to an output/input field; its position in the output record is not necessarily the same as in the input record.

All three types of fields can be used in the same display screen format.

Note: The record defined for a TRANSACTION file must be large enough to accept the data defined for the screen.

In addition to specifying the type of field, you can specify a number of other attributes of each field. One attribute that you can specify is what type of data the field contains. The types of data that can be specified are:

- Alphabetic only.

- Numeric only. If special characters are entered in a numeric field, the data read by the COBOL program may not be as expected. The program uses only the digit portion of each character, forcing the zone portion to hex F (except for the sign position).

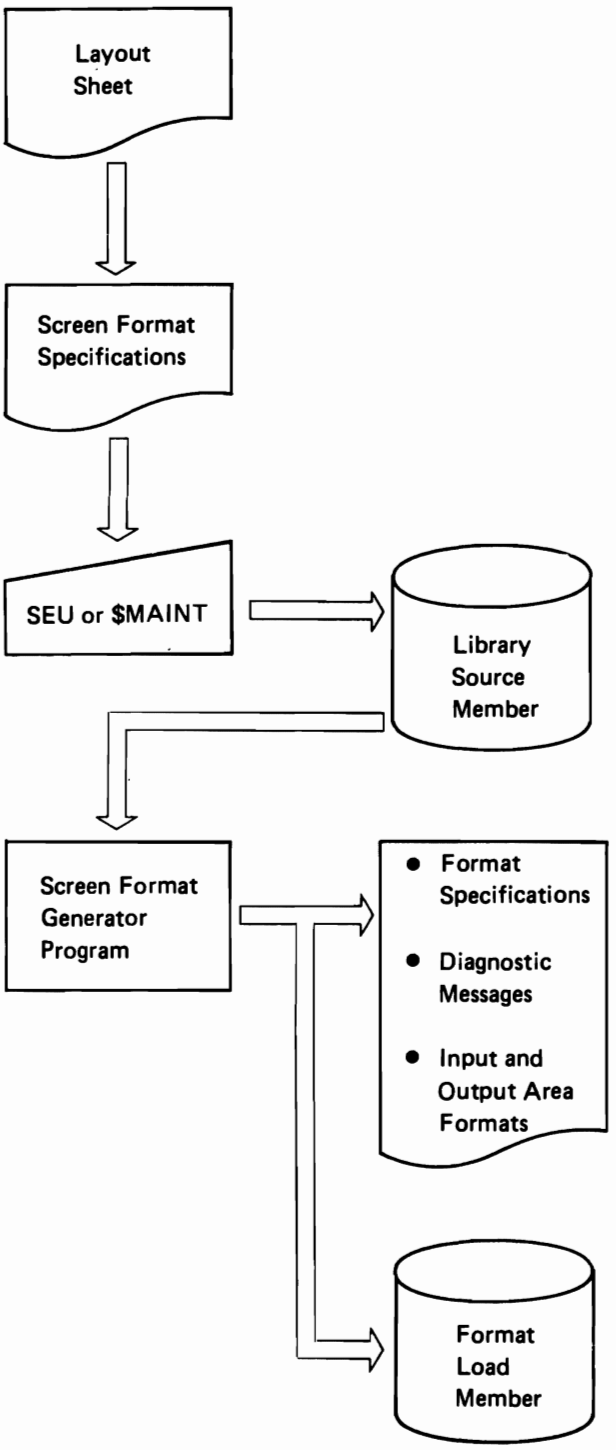
- Alphanumeric.

- Signed numeric (the last position of each field contains a sign). For signed numeric fields, the length of the field specified on the display screen format specifications must be one more than the length specified in the COBOL program. This allows the sign to be displayed as a separate character on the screen.

Among the other attributes that can be specified are the display intensity, whether the field is a blinking field, and whether data must be entered into the field before the record can be returned to the program. These attributes can be made conditional by specifying an indicator in the appropriate columns of the D specifications. If the indicator is on in the program, the field has the specified attribute. If the indicator is off in the program, the field does not have the attribute.

The S specifications define the attributes of the display screen format as a whole. On the S specifications you can specify the number of lines to be cleared before the format is written on the display screen, whether or not to sound an alarm when this format is displayed, and the starting line of the display format. For a complete description of what can be specified on the S and D specifications, see the $SFGR utility program in the *System Support Reference Manual.* For a summary of the possible entries, see Appendix I in this manual.

When you have completed the display screen format specifications, use the $MAINT utility program or SEU to enter the specifications into a source member. Then run the screen format generator program ($SFGR utility program) to process the source specifications. The name of the format member must appear on the ASSIGN clause of the File Control entry. For an example of a File Control entry that associates a screen format member with a TRANSACTION file, see 1 in Figure 7-13. Figure 7-4 shows the steps for creating a format load member.



1. Arrange all fields on the layout sheet, just as they will appear on the display screen.

2. Use the completed layout sheet as a guide for filling out the two parts of the display screen format specifications.

3. Use SEU or $MAINT to transfer the specifications to a library source member.

4. Run the screen format generator utility program ($SFGR) by entering the required control statements or by running the FORMAT procedure. The screen format generator program produces the following output:
   - List of source specifications
   - Diagnostic messages
   - List of information about the input and output records for the format
   - The display screen format (or formats) in the specified load member

Figure 7-4. Steps for Creating a Display Screen Format

End-of-File Considerations

End-of-file occurs for an SRT or MRT TRANSACTION file program when there are no attached display stations or SSP-ICF sessions for which there is an outstanding invited input operation. A write operation must precede each read operation in order for the display station to become input-invited. This is required if a subsequent read operation is to handle the next input record from that display station.

End-of-file occurs for an NEP TRANSACTION file program when there are no attached display stations or SSP-ICF sessions and the operator enters a STOP SYSTEM command.

When end-of-file occurs, an indication passes back to the COBOL program. This indication causes the AT END condition on the READ operation to the TRANSACTION file or the user exception/error procedure to be executed (whichever is specified). The AT END condition allows the programmer to control the program flow when end-of-file is reached. (For more information on the AT END condition, see the READ statement in Chapter 5.)

Note: If the AT END condition is not specified, the user exception/error procedure is executed. If no exception/error procedure is specified, the program halts and an error message that forces program termination is displayed when end-of-file is reached.

SPECIAL DISPLAY SCREEN FORMAT CONSIDERATIONS

Overriding Fields in a Format

An override operation allows you to override fields in a format when you redisplay that same format. To perform an override function, you must specify an indicator in the override field, columns 33 and 34 of the S specification. An override operation is peformed if the indicator is on when the format is displayed (see Figure 7-5). The entire format is displayed if the indicator is off when the format is displayed.

During an override operation (the indicator in the override field, columns 33 and 34, is on) one of the following occurs:

- Any field that has an indicator specified in the output field (columns 23 and 24 of the D specification) remains unchanged if the indicator is off. If data was keyed into the field, that data is not changed. Any field that had Y, N, or blank specified in columns 23 and 24 is also not changed.

- Any field that has an indicator specified in the output field (columns 23 and 24) is displayed with data from the COBOL program when the indicator is on. Any data that was keyed into the field by the operator is written over and lost. Output information is displayed from the same locations in the output record area as for a normal display.

For all fields, the use of indicator-controlled attributes such as highlight or reverse image is determined by the state of that indicator. All field attributes that are not controlled by indicators are unchanged.

For example, you may want to override fields in a display if the operator keys incorrect data into a field. To do this, specify an indicator in columns 33 and 34 of the S specification, which allows that format to be overridden. If the operator keys incorrect data into a field, you can set on the indicator specified in columns 33 and 34 and redisplay the format. If the indicator specified in columns 23 and 24 of the D specification for the field is off, the incorrect data is redisplayed unchanged. If this indicator is on, data from the COBOL program is displayed in place of the incorrect data. The attributes specified for the field remain the same.

		Indicator in Columns 33 and 34 of the S Specification	
		OFF	**ON**
Indicator in Columns 23 and 24 of the D Specification	**OFF**	Output data comes from the D specification (columns 57 through 79)	No change occurs to data on the screen
	ON	Output data comes from the COBOL program	Output data comes from the COBOL program

Figure 7-5. Effect of Indicators on Output Data During an Override Operation

Read Under Format

A read under format allows one program in a procedure to display a format and the next program in the procedure to read it. A program displays the format using a normal output operation and then either goes to end of job or releases the display station. While the second program is initiating, the operator keys data into the displayed format. When the operator presses the Enter/Rec Adv key, the input information from the display is sent to the second program.

The following steps occur in a read under format:

1. With a normal output operation, the first program displays a format at the display station.

2. The first program goes to end of job (if the program is an SRT program) or releases the requesting display station (if the program is an MRT program). The display station is released when the DROP operation is executed in the Procedure Division, or when the program goes to end of job.

3. The second program is initiated. (Data should not be passed to the second program from an INCLUDE OCL statement. For more information on the INCLUDE OCL statement, see *INCLUDE Statement* in Chapter 1 of the *System Support Reference Manual*.)

4. The second program performs a normal TRANSACTION file read operation.

The PROMPT OCL statement also causes a read under format operation to take place. The PROMPT OCL statement cannot be used in an MRT program, or a program run from the input job queue. For more information on the PROMPT OCL statement, see *PROMPT Statement* in the *System Support Reference Manual*.

Command Keys

The display screen format S specification allows you to enable specific command keys for your program (if column 28 of the S specification is blank, all command keys are enabled for a TRANSACTION file program). All command keys that are not enabled are considered to be disabled. An error message is displayed when a disabled command key is pressed. The operator must press the Error Reset key to unlock the keyboard, and may then press the correct command key. For more information on enabling and disabling command keys, see the summary of the display screen format S specification in Appendix I or see *Display Screen Format Specifications* in the *System Support Reference Manual.*

Command keys do not set on indicators in the COBOL program. When a command key is pressed, the number corresponding to the numeric value of the command key is placed into the Function Key field of the CONTROL-AREA. (For example, if command key 7 is pressed, a 07 is placed in the Function Key field.) It is the responsibility of the COBOL program logic to test for command key values in the CONTROL-AREA Function Key field (for more information on the format of the CONTROL-AREA, see *CONTROL-AREA Clause* in this chapter).

ENVIRONMENT DIVISION

SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph relates IBM-specified function-names to user-specified mnemonic-names. The function-names SYSTEM-SHUTDOWN, LOCAL-DATA, and ATTRIBUTE-DATA have been added for use with TRANSACTION file processing. The UPSI switch processing has been expanded. The format and description of the SPECIAL-NAMES paragraphs are described under *Configuration Section* in Chapter 3. The additional requirements for transaction processing are discussed in the following paragraphs.

Function-Name-1 Clause

LOCAL-DATA and ATTRIBUTE-DATA can be specified in the function-name-1 clause. The associated mnemonic-name is formed according to the rules for a user-specified name and is required to contain at least one alphabetic character. The following paragraphs describe the actions associated with LOCAL-DATA and ATTRIBUTE-DATA.

LOCAL-DATA: The mnemonic-name associated with LOCAL-DATA can be referenced by either an ACCEPT or a DISPLAY statement which references an attached requestor. An ACCEPT or a DISPLAY statement is issued to retrieve data from, or store data in, a system managed area that provides communications among programs that are executed sequentially within a display station session.

ATTRIBUTE-DATA: The mnenomic-name associated with ATTRIBUTE-DATA can be referenced only in an ACCEPT statement. The reference causes an attribute-record that is related to an identified device to be input to the data item coded in the ACCEPT statement. Attribute-records are described in Figure 7-6 and 7-7.

```
SPECIAL-NAMES.
        ATTRIBUTE-DATA IS ATTRIBUTES.
            .
            .
            .
01      TERMINAL-ATTRIBUTES.
        ATTRIBUTES OF A TERMINAL RETRIEVED BY ACCEPT
        02 TERMINAL-TYPE                 PICX.
        02 TERMINAL-SIZE                 PICX.
        02 TERMINAL-LOCATION             PICX.
        02 TERMINAL-ON-OFF-LINE          PICX.
        02 TERMINAL-ALLOCATION-STATUS    PICX.
        02 TERMINAL-INPUT-STATUS         PICX.
        02 TERMINAL-DATA-STATUS          PICX.
        02 TERMINAL-INQUIRY-STATUS       PICX.
            .
            .
            .
        ACCEPT TERMINAL-ATTRIBUTES FROM ATTRIBUTES.
```

Definition of Values Returned in Attribute Record for a Work Station:

Byte 1 Type of work station terminal
- D Display
- N Printer
- 2 Unknown

Byte 2 Device size
- 1 1920-character display screen
- 2 960-character display screen

Byte 3 Local or remote device
- L Local
- R Remote

Byte 4 Online or offline
- O Online
- F Offline

Byte 5 Device allocation status
- A Allocated to this program
- E Allocated to another program
- V Unallocated available (may be acquired)
- N Unallocated not available (command terminal in command mode)
- U Unknown

Byte 6 Input status
- Y Input is allowed (an invite input is outstanding)
- N Input is not allowed

Figure 7-6 (Part 1 of 2). Attribute-Record for a Work Station

SPECIAL-NAMES.
ATTRIBUTE-DATA IS ATTRIBUTES. (Continued)

Definition of Values Returned in Attribute Record for a Work Station (Continued):

Byte 7 Data status
- Y Data is currently pending (Enter or a Command/Function key has been pressed)
- N Data is not pending

Byte 8 Inquiry Status
- Y Display station is in inquiry mode
- N Display station is not in inquiry mode

Byte 9 Display type
- A Alphanumeric or Katakana
- I Ideographic

Byte 10 Keyboard type
- A Alphanumeric or Katakana
- I Ideographic

Byte 11 Sign-on mode
- A Alphanumeric or Katakana
- I Ideographic

Bytes 12-16 Reserved

Figure 7-6 (Part 2 of 2). Attribute-Record for a Work Station

```
SPECIAL-NAMES.
        ATTRIBUTE-DATA IS ATTRIBUTES.
                .
                .
                .
01      SESSION-ATTRIBUTES.
        ATTRIBUTES OF A SESSION RETRIEVED BY ACCEPT
        02 SESSION-STATUS                PICX.
        02 INVITE-STATUS                 PICX.
        02 SESSION-NAME                  PICX(6).
                .
                .
                .
        ACCEPT SESSION-ATTRIBUTES FROM ATTRIBUTES.
```

Definition of Value Returned in Attribute Record for an SSP-ICF Session:

Byte 1

- A The SSP-ICF session has a SESSION OCL statement but has not yet been acquired.
- C The SSP-ICF session has been acquired.
- R The SSP-ICF session is an evoked session; that is, an evoke has been issued by the remote system, and the transaction has not yet ended.

Byte 2

- N This SSP-ICF session has not been invited.
- I This SSP-ICF session has been invited, but no data has been received.
- O This SSP-ICF session has been invited; the data is ready.

Bytes 3 through 8

The 6-character configuration name associated with the SSP-ICF session.

Figure 7-7. Attribute-Record for an SSP-ICF Session

Function-Name-2 Clause

The format of the SPECIAL-NAMES entry for the UPSI switches is as described under *SPECIAL-NAMES Paragraph* in Chapter 3. The processing of the switch is expanded as follows:

UPSI (User Program Status Indicator) switch: UPSI-0 through UPSI-7 define external switches that are set and tested externally by the command language, or internally by a COBOL source statement. UPSI switches are tested by an IF statement. The current UPSI values are retrieved from the system and the test is performed against these switch settings.

When UPSI switches are updated by a SET statement, the current UPSI values are retrieved from the system and these switch settings are updated as specified by the SET. The new switch settings are then returned to the system for future references.

Note: A separate copy of the settings of the UPSI switches is kept for each requestor of an MRT program. When a requestor gains control of the program, the UPSI switches are automatically set to the values stored for that requestor.

SYSTEM-SHUTDOWN: SYSTEM-SHUTDOWN is an internal switch that is set to the ON status when the system operator causes the system to be in a shutdown pending state. The associated ON or OFF condition-names can be referenced in any conditional expression.

File Control Entry

The TRANSACTION file must be named by a file control entry in the FILE-CONTROL paragraph. This entry also specifies other information related to the file. There can be only one TRANSACTION file in a run unit.

Format

```
SELECT file-name

    ASSIGN TO assignment-name

    ORGANIZATION IS TRANSACTION

[ FILE STATUS IS data-name-1 [ , data-name-4 ] ]

[ACCESS MODE IS SEQUENTIAL ]

[CONTROL-AREA IS data-name-5 ].
```

The following File Control entry defines a TRANSACTION file that is associated with a screen format member named CBTESTFM. CBTESTFM can have a maximum of 32 screen formats in it. The FILE STATUS area is labeled RETN-CODE and the CONTROL AREA is labeled CNTL-AREA.

```
FILE-CONTROL.
    SELECT SFILE ASSIGN TO WORKSTATION-CBTESTFM-32
    ORGANIZATION IS TRANSACTION
    FILE STATUS IS RETN-CODE
    CONTROL-AREA IS CNTL-AREA.
```

ASSIGN Clause

The ASSIGN clause associates the TRANSACTION file with devices through the use of the assignment-name. Assignment-name has the following structure:

```
Type [-Name        ]
     [-Name-Formats]
```

The value for each field is as follows:

Type:	WORKSTATION
Name:	1- to 8-character name that specifies the external name of the SFGR generated load member that contains the screen formats. This field is not required if the file is to be used with SSP-ICF sessions only.
Formats:	A two-digit numeric value that is equal to or greater than the number of formats in the SFGR load member referenced in the name field. The maximum value and the default value for the number of formats is the same, 32.

The COBOL compiler constructs an internal table to hold information about each format. The value specified by Formats determines how many 16-byte entries are in the internal table. The actual number of formats in your program does not influence the number of entries in this internal table.

ORGANIZATION Clause

The ORGANIZATION clause specifies the logical structure of a file. TRANSACTION organization signifies user-controlled input and output of records.

ACCESS MODE Clause

The ACCESS MODE clause is described under *FILE-CONTROL Paragraph* in Chapter 3.

FILE STATUS Clause

The FILE STATUS clause defines the area that contains status keys for a TRANSACTION file. Data-name-1 identifies the ANSI COBOL status keys (status key 1 and status key 2). The possible combinations of status key 1 and status key 2 are shown in Appendix H. Data-name-4 identifies the extended FILE STATUS, which contains display station and SSP-ICF return codes. The first 2 bytes of the extended FILE STATUS contain the major return code; the second 2 bytes contain the minor return code. Extended return codes for the Interactive Communications Feature are documented in the *Interactive Communications Feature Reference Manual*. Figure 7-8 correlates the ANSI status keys and the extended FILE STATUS return codes.

ANSI Status Key	Major Return Code	Explanation
00	00	Normal completion (operation was successful).
	03	No data received.
	08	Acquire of owned session.
01	01	New requestor.
10	11	Accept rejected, no invites outstanding.
30	80	Permanent error during session. The session has been terminated. The subsystem must be enabled before more sessions can be started.
92	81	Session error. Session is terminated, and must be reacquired to be used.
9A	02	Stop system or disable pending.
9C	18	Acquire failed, temporarily not available.
	82	Acquire failed.
9D	24	Display station released by operator.
9E	28	SRT attempting to communicate on a session it previously released.
9F	32	Acquire failed, unauthorized user.
9G	34	Input rejected, buffer too small.
9H	38	Acquire failed, not waitable.
9I	04	Output exception
	44	Stop invite failed, data available.
9N	83	Session error. Session is still active.

Figure 7-8. ANSI Status Keys and Their Corresponding Major Return Codes

CONTROL-AREA Clause

The CONTROL-AREA clause specifies the 12-byte data item which receives feed-back information upon completion of a TRANSACTION file input operation. The information is in the fixed format as shown in Figure 7-9. Each item in the feed-back area is described as follows:

Bytes 1-2 (Function Key): The Function Key is a two-digit numeric or alphanumeric data item inserted by the work station interface that identifies which function key the operator pressed to initiate the transaction. The codes are as follows:

00	Enter key
01-24	COMMAND keys 1 through 24
90	ROLL UP key
91	ROLL DOWN key
99	Undefined

Bytes 3-4 (Terminal ID): The symbolic identification of a device. The Terminal ID must be defined as a two-byte alphanumeric data item.

Bytes 5-12 (Reserved): Reserved for future use.

```
FILE-CONTROL.
SELECT SCREEN-FILE ASSIGN TO WORKSTATION-MYFMTS-12
  ORGANIZATION IS TRANSACTION
  CONTROL-AREA IS TRANSACTION-CONTROL-AREA.
          .
          .
          .
  WORKING-STORAGE SECTION.
01  TRANSACTION-CONTROL-AREA.
          03 FUNCTION-KEY              PIC 99.
          03 TERMINAL-ID               PIC X(2).
          03 RESERVED                  PIC X(8).
```

Figure 7-9. Sample FILE-CONTROL Paragraph and CONTROL-AREA

File Description Entry

A file description entry consists of a level indicator (FD), a file-name, and a series of independent clauses. For a TRANSACTION file, the independent clauses allowed are the RECORD CONTAINS clause, the LABEL RECORDS clause, and the DATA RECORDS clause. Only the LABEL RECORDS clause is required.

Format

```
FD file-name
   [ RECORD CONTAINS [ integer-3 TO ] integer-4 CHARACTERS ]
   LABEL { RECORDS ARE } OMITTED
         { RECORD IS   }
   [ DATA { RECORD IS  } data-name-3 [ data-name-4 ] ... ] .
          { RECORDS ARE }
```

The following file description entry might be used for a TRANSACTION file. The file description entry defines a record 401 characters long. The name of the record area is DS-REC.

```
FD    DSFILE
      RECORD CONTAINS 401 CHARACTERS
      LABEL RECORDS ARE OMITTED
      DATA RECORD IS DS-REC.
```

The LABEL RECORDS clause specifies whether or not labels are present. Label records must be omitted for a TRANSACTION file. This clause is required in every file description entry.

The RECORD CONTAINS clause and the DATA RECORDS clause are described under *RECORDS CONTAINS Clause* and *DATA RECORDS Clause* in Chapter 4. The record definition must be large enough to hold the largest record defined by the SFGR formats or SSP-ICF records processed by the program.

Boolean Data Facilities

Boolean data provides a means of modifying and passing the values of the indicators associated with the display screen formats. A Boolean value of 0 is the indicator's off status while a Boolean value of 1 is the indicator's on status.

A Boolean literal contains a single 0 or 1 enclosed in quotes and is immediately preceded by an identifying B. The Boolean literal is defined as either B'0' or B'1'. A Boolean character occupies one byte. The figurative constant ZERO can be used as a Boolean literal, and the reserved word ALL is valid with a Boolean literal. The Boolean ZERO is the fill character for Boolean data.

Note: A hex value of F0 is considered an off value for Boolean data types. A hex value of F1 is considered an on value for Boolean data types.

Data Description Entry–Boolean Data

The data description entry for Boolean data can contain the following clauses.

Format

level-number { data-name / FILLER } clause

[REDEFINES clause]

[USAGE clause]

[OCCURS clause]

[SYNCHRONIZED clause]

[JUSTIFIED clause]

[VALUE clause]

[PICTURE clause

[INDICATOR clause]

Special Considerations

The special considerations for the clauses used with Boolean data follow. All other rules for clauses are the same as those for other data as described under *Data Description Entry* in Chapter 4.

PICTURE Clause: An elementary Boolean data-name is defined by a PICTURE containing a single 1.

USAGE: USAGE must be defined implicitly or explicitly as DISPLAY.

OCCURS Clause: The ASCENDING/DESCENDING key is not valid for Boolean data items. The OCCURS clause has special consideration when used with the INDICATOR clause as described later in this chapter.

If the OCCURS clause and INDICATOR clause are both specified at an elementary level, a table of Boolean data items is defined. Each element in the table corresponds to an external indicator. (For an example of a table of Boolean data items, see *INDICATOR Clause* below.)

VALUE Clause: The VALUE clause specifies the initial content of a Boolean data item. The format is:

[VALUE IS boolean-literal].

INDICATOR Clause: The INDICATOR clause is used to associate an SFGR indicator number with a Boolean data-item.

Format

INDICATOR integer

Integer must be greater than or equal to 1, and less than or equal to 99.

The INDICATOR clause must be specified at an elementary level only.

Since an indicator can contain only a value of zero or one, it may be associated only with a Boolean data-item.

OCCURS Clause with the INDICATOR Clause: If the OCCURS clause and the INDICATOR clause are both specified at an elementary level, a table of Boolean data items is defined with each element in the table corresponding to an external indicator. The first element in the table corresponds to the indicator number specified in the INDICATOR clause, the second element corresponds to the indicator which sequentially follows the indicator specified by the INDICATOR clause.

For example, if the following is coded:

```
07 SWITCHES  PIC 1 OCCURS 10 TIMES
             INDICATOR 16.
```

then:

SWITCHES (1) corresponds to SFGR indicator 16,
SWITCHES (2) corresponds to SFGR indicator 17,...
SWITCHES (10) corresponds to SFGR indicator 25.

Note: A Boolean data item associated with an SFGR indicator can be used to REDEFINE another Boolean data item associated with a different SFGR indicator. However, if you specify a group-name that contains both these Boolean data items on a WRITE statement that has the INDICATORS phrase specified, undesirable results can occur.

EXCEPTION/ERROR Declaratives

The EXCEPTION/ERROR Declarative specifies procedures for input-output error handling that are in addition to the standard procedures provided by the input-output control system.

Format

```
USE AFTER STANDARD {EXCEPTION}
                   {ERROR    }
    PROCEDURE ON {file-name-1 [file-name-2] ...}.
                 {I-O                       }
```

In an EXCEPTION/ERROR Declarative for TRANSACTION files only the file-name or I-O options are allowed. All other options and rules are the same as those for any EXCEPTION/ERROR Declarative for any file. The options and rules are described under EXCEPTION/ERROR Declaratives in Chapter 5.

ACCEPT Statement

The ACCEPT statement causes low-volume data to be made available to the specified data item.

Format

```
ACCEPT identifier-1 FROM mnemonic-name
[FOR {identifier-2}]
     {literal     }
```

If the mnemonic-name is associated with LOCAL-DATA, the 256-byte local data area associated with the requestor terminal is moved into identifier-1.

If mnemonic-name is associated with ATTRIBUTE-DATA, identifier-1 must describe an attribute data record. (Attribute data records are described under SPECIAL-NAMES paragraph in this chapter.) The attributes of the specified symbolic ID are moved into identifier-1.

The move into identifier-1 for both LOCAL-DATA and ATTRIBUTE-DATA takes place according to the rules for the MOVE statement for an alphanumeric group move without the CORRESPONDING option.

FOR Option

The FOR option is allowed only when mnemonic-name is associated with either ATTRIBUTE-DATA or LOCAL-DATA. Literal or the contents of identifier-2 is the symbolic ID for which data is retrieved. A symbolic ID of blanks retrieves the attributes of the requestor for batch jobs.

If the mnemonic-name is associated with either SYSTEM-CONSOLE or REQUESTOR, the FOR option is not valid. For additional information refer to *ACCEPT Statement* in Chapter 5.

ACQUIRE Statement

The ACQUIRE statement attaches a device to the TRANSACTION file.

Format

ACQUIRE { literal / identifier } FOR file-name

The following ACQUIRE statement can be used to attach a device to a program.

```
ACQUIRE DS-CURRENT-ID FOR WSFILE.
```

The value of literal or identifier specifies the symbolic identification of a device that is to be associated with file-name. In order to be acquired, a display station must be in stand-by mode. In order to acquire an SSP-ICF session, it must be specified in the SESSION OCL statement for the job step.

If literal is specified, it must be a two-character alphanumeric literal. If identifier is specified, it must refer to a two-character alphanumeric data item.

File-name must refer to a file whose organization is TRANSACTION.

CLOSE Statement

The CLOSE statement terminates the processing of the TRANSACTION file. All devices that had been acquired for the TRANSACTION file are released when the file is closed. Once the TRANSACTION file has been closed, it cannot be reopened in the same program.

Format

CLOSE file-name-1 [WITH LOCK]

The following CLOSE statement closes a TRANSACTION file named DSFILE with lock.

```
CLOSE DSFILE WITH LOCK.
```

If a CLOSE statement is executed for the TRANSACTION file, no other statements that reference that file can be executed. A TRANSACTION file is locked when closed.

For general information about closing a file, refer to *CLOSE Statement* in Chapter 5.

Programming Note: For TRANSACTION files, the WITH LOCK option of the CLOSE statement should be specified for documentation.

DISPLAY Statement

The DISPLAY statement causes low-volume data to be transferred to an appropriate hardware device.

Format

```
DISPLAY {identifier-1} [,identifier-2] ... UPON mnemonic-name
        {literal-1   } [,literal-2   ]
    [FOR {literal-3   }]
         {identifier-3}
```

This format of the DISPLAY statement is applicable when mnemonic-name is associated with the system name LOCAL-DATA. If mnemonic-name is associated with system-name of either SYSTEM-CONSOLE or REQUESTOR, refer to *Display Statement* in Chapter 5.

Literal-1 or the content of identifier-1 is written to the 256-byte local data area associated with the requestor.

Literal-3 or the contents of identifier-3 must be the valid symbolic ID of an attached requestor.

DROP Statement

The DROP statement releases a device from its association with the TRANSACTION file.

Format

```
DROP {literal   } FROM file-name
     {identifier}
```

The following DROP statement releases a device from the TRANSACTION file, DSFILE.

```
DROP DS-CURRENT-ID FROM DSFILE.
```

The value of literal or identifier specifies the symbolic identification of the attached device that is to be released.

If literal is specified, it must be a two-character alphanumeric literal. If identifier is specified, it must refer to a two-character alphanumeric data item.

The DROP statement can only be used with a TRANSACTION file. At the end of program execution, all attached devices are implicitly released.

OPEN Statement

The OPEN statement initiates the processing of files. Only one file with organization TRANSACTION is allowed per program.

Format

```
OPEN I-O file-name-1 [file-name-2] ...
```

A TRANSACTION file must be opened with the I-O phrase.

A TRANSACTION file can only be opened once in a program.

For general information about opening files, refer to *OPEN Statement* in Chapter 5.

The following OPEN statement opens the TRANSACTION file, DSFILE, in I/O mode.

```
OPEN I-O DSFILE.
```

READ Statement

The READ statement makes a record available from the TRANSACTION file or allows a new device to be attached.

The TRANSACTION file must be open in the I-O mode at the time the READ statement is executed.

Format

READ file-name RECORD

[INTO identifier-1] [TERMINAL IS {identifier-2 / literal-1}]

[NO DATA imperative-statement-1]

[AT END imperative-statement-2]

The following READ statement reads from the TRANSACTION file, DSFILE. The record is read from the device whose ID is presently in DS-CURRENT-ID. This statement specifies both the NO DATA option and the AT END condition, which are described in the following write up.

```
READDSFILE INTO DATA-AREA TERMINAL
    DS-CURRENT-ID, NO DATA GO TO
    NO-INPUT, AT END GO TO EOJ-EXIT.
```

Upon successful execution of the READ statement, the terminal-id and function key fields of the CONTROL-AREA, if present, are filled in.

Programming Note: A write operation must follow each read operation in order for the display station to become input-invited. This is required if a subsequent read operation is to handle the next input record from that display station.

TERMINAL Option

The record to be made available by a READ statement is determined as follows:

- If the TERMINAL option is specified, literal-1 or the contents of identifier-2 must be the symbolic ID of an attached device associated with the TRANSACTION file. The data record is made available from the device specified.

- If the TERMINAL option is omitted, the defaults are:
  - If a single device is attached to the file, the default is that device.
  - If multiple devices are attached to the file, there is no default. The data record made available is the first record input from any attached device.

Programming Note 1: Use of the TERMINAL option forces the next input to come from the specified device. When identifier-2 contains blanks, the READ statement is executed as though the TERMINAL was not specified.

Programming Note 2: The TERMINAL option should usually not be specified for an MRT program. When the TERMINAL option is specified, program execution halts until a record is available from the device specified by the TERMINAL option. This can cause the other users of the program to wait while the record is being entered at the specified device. If you must specify the TERMINAL option, you should also specify the NO DATA option to allow processing to continue if a record is not currently available.

NO DATA Option

When the NO DATA option is specified, the imperative-statement specified is executed if a record cannot immediately be read from the specified device. This allows your program to continue processing, rather than waiting for the operator to complete entering a record.

When the NO DATA option is not specified, a record is available to the object program prior to the execution of any statement following the READ statement.

AT END Option

The AT END condition occurs when there are no attached devices for which an input operation is currently invited and the program is not a NEP. The AT END condition occurs for a NEP when there are no attached devices and the system operator has entered a STOP SYSTEM command.

Input is implicitly invited with each WRITE statement but can be suppressed by an option on the SFGR format or selected SSP-ICF predefined formats. When the AT END condition occurs, the READ statement is unsuccesful and imperative-statement-2 is executed.

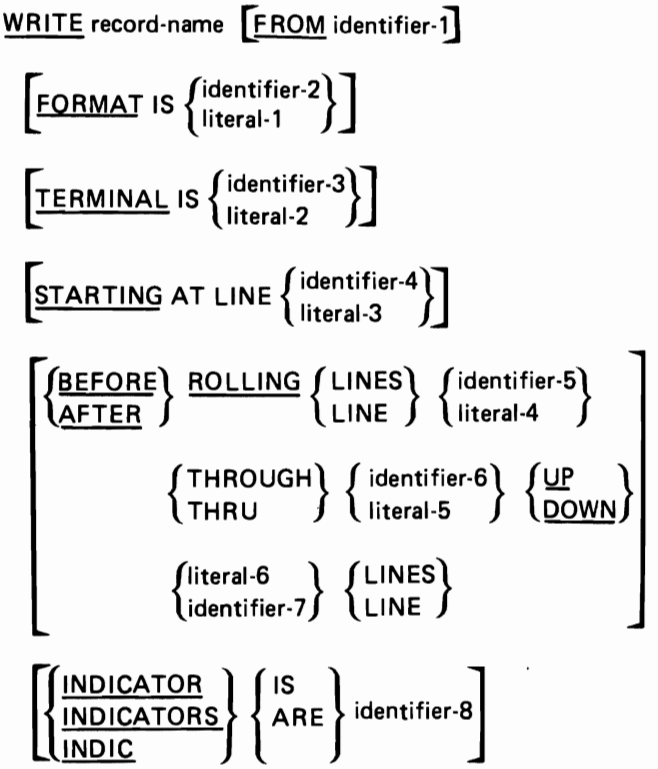
Programming Note: The AT END condition does not occur when the READ statement includes the TERMINAL option.

For further discussion of the READ Statement and the INTO and AT END options, see *READ Statement* in Chapter 5.

WRITE Statement

The WRITE statement releases a logical record to the TRANSACTION file. This file must be opened in the I-O mode at the time the WRITE statement is executed.

Format



The following WRITE statement writes a record (DS-REC) and a format, FMT(INDX), to the display station whose ID is presently in DS-CURRENT-ID. The information is written on the display screen after rolling the present information up.

```
WRITE  DS-REC FORMAT FMT(INDX) TERMINAL
       DS-CURRENT-ID, AFTER ROLLING
       DS-START-LINE-FOR-ROLL THRU
       DS-END-LINE-FOR-ROLL UP WS-
       NUMBER-OF-LINES-TO-ROLL.
```

Requirements for record-name and the FROM option are discussed under *Write Statement* in Chapter 5.

Literal-1 and literal-2 must be nonnumeric. Literal-3, literal-4, literal-5, and literal-6 must be numeric.

Identifier-2 and identifier-3 must be alphanumeric data items. Identifier-4, identifier-5, identifier-6, and identifier-7 must be elementary numeric items. Identifier-8 must be one of the following:

- An elementary Boolean data item specified without the OCCURS clause.
- A group item that has elementary Boolean data items subordinate to it. The group item should not contain an indicator data item that redefines another indicator data item within that group item.

FORMAT Option

The record specified by the record-name is sent to the specified destination using the named format. A format must be specified for the first WRITE verb executed. If subsequent WRITE operations do not include a FORMAT option, the most recently used format is used. The FORMAT option contains the name of the screen format used when data is written to the work station. This format must be in the format load member. The member name is specified as part of the assignment-name in the ASSIGN clause for the TRANSACTION file.

Writing to the Error Line: If the contents of identifier-2 or the value of literal-1 is ERRLINE, System/34 COBOL writes to the error line of the display station instead of writing a format. A read operation should follow the write operation. This locks the display station to keep the error line from being overwritten. The operator presses the Reset key to return to entry mode. A write to the error line causes the last line of output on the screen to be saved, and the output record to replace the bottom line on the screen. When the operator presses the RESET key, the last line of output on the screen reappears.

Note: If a WRITE statement specifies a format name of ERRLINE, the ROLLING BEFORE and AFTER options cannot be specified.

Interactive Communications Feature: Special format names are recognized by Data Management that provide the COBOL user SSP-ICF functions. The uses of these special format names and the functions of ICF are described in the *ICF Reference Manual*. The system-defined special format names begin with two dollar signs ($$), as shown below. (You should not begin your display screen format names with $$.)

$$EVOK	$$TIMER
$$EVOKNI	$$NRSP
$$EVOKET	$$NRSPNI
$$SEND	$$CANL
$$SENDNI	$$CANLNI
$$SENDE	$$EOS
$$SENDET	$$PTPUT
$$SENDFM	$$RCD
$$SENDNF	$$PTINV
$$FAIL	

TERMINAL Option

The TERMINAL phrase specifies the destination to which the record is to be sent. If the TERMINAL option is not specified for a single device file, that device is the destination. If the TERMINAL option is not specified for a multiple device file, the most recent source or destination identifier is used as the destination.

STARTING Option

The STARTING phrase contains the starting line number for screen formats that use the variable start line option. If the value of this element is less than 01 or greater than 24, a value of 01 is assumed. If the screen format does not specify this option, this value is ignored.

ROLLING Option

The ROLLING option allows you to move the data presently displayed on the display screen. All or part of the data on the screen can be rolled up or down. The lines vacated by the rolled data are cleared, and can have another screen format written into them.

Rolling is specified on the WRITE statement that is writing a new format to the display screen. The number of lines you want to roll, how many lines you want to roll these lines, and whether the roll operation is up or down must be specified.

Rolling ignores field attributes. The data is rolled exactly as it appears on the display screen. Its associated attributes (whether it is an input field, an input/output field, and so on) are not rolled with the data and are lost. Therefore, after a field has been rolled, it can no longer be input capable. If you specify ROLLING BEFORE and part of the new format is moved, this portion of the format is not input capable.

Note: The value specified by identifier-5 (or literal-4) must be less than the value specified by identifier-6 (or literal-5).

Figure 7-10 shows how rolling might be used. An initial screen format, FMT-1 has been written on the display screen. The program has processed this screen format, and is now ready to write the next screen format, FMT-2, to the display screen. Part of FMT-1 is to be rolled down two lines before FMT-2 is written to the screen.

The following WRITE statement causes part of FMT-1 to be rolled down two lines and writes FMT-2 to the display screen:

```
WRITE SCREEN-RECORD, FORMAT FMT-2
    AFTER ROLLING LINES 14 THRU 19 DOWN 2
    LINES
```

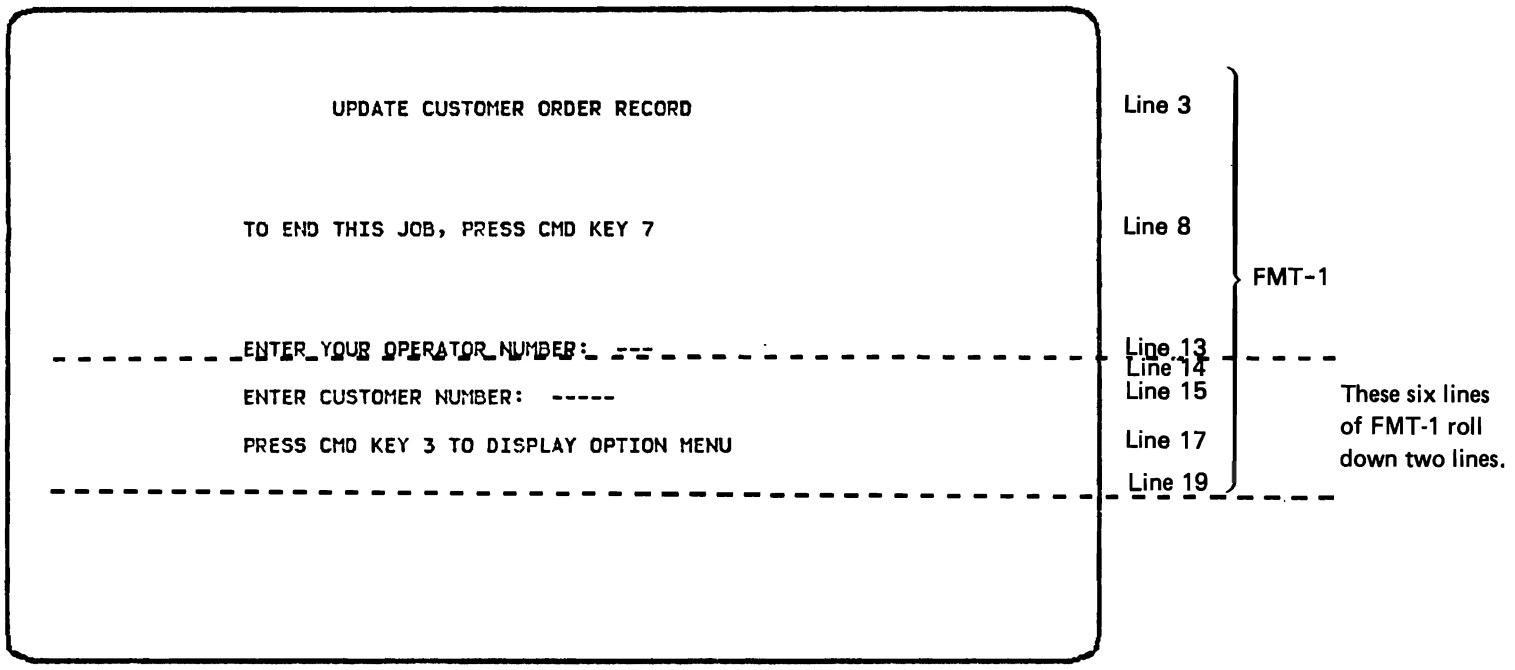
When this WRITE statement is executed, the following occurs:

- The contents of lines 14 through 19 are rolled down two lines, one line at a time.
- The contents of lines 18 and 19 (blank in this example) are rolled off the window and no longer exist.
- The area vacated by the roll operation is cleared. In this case, only lines 14 and 15 are cleared.
- After the rolling operation takes place, FMT-2 is written to the display screen. This format does not clear the display screen (number of lines to clear, columns 19 and 20 of the S specification for SFGR are blank). FMT-2 is written both in the area vacated by the roll operation, and outside it in an area still containing data from FMT-1. FMT-2 is written over the data left from FMT-1. When the contents of the display screen are returned to the program, only the input capable fields from FMT-2 are returned.

INDICATOR Option

The INDICATOR phrase specifies the name of an area that contains indicator information. DSDM ignores provided indicators that are not specified on the SFGR format. Indicators not provided in the indicator area are considered by DSDM to be off.

DISPLAY BEFORE ROLL



DISPLAY AFTER THE ROLL DOWN OPERATION

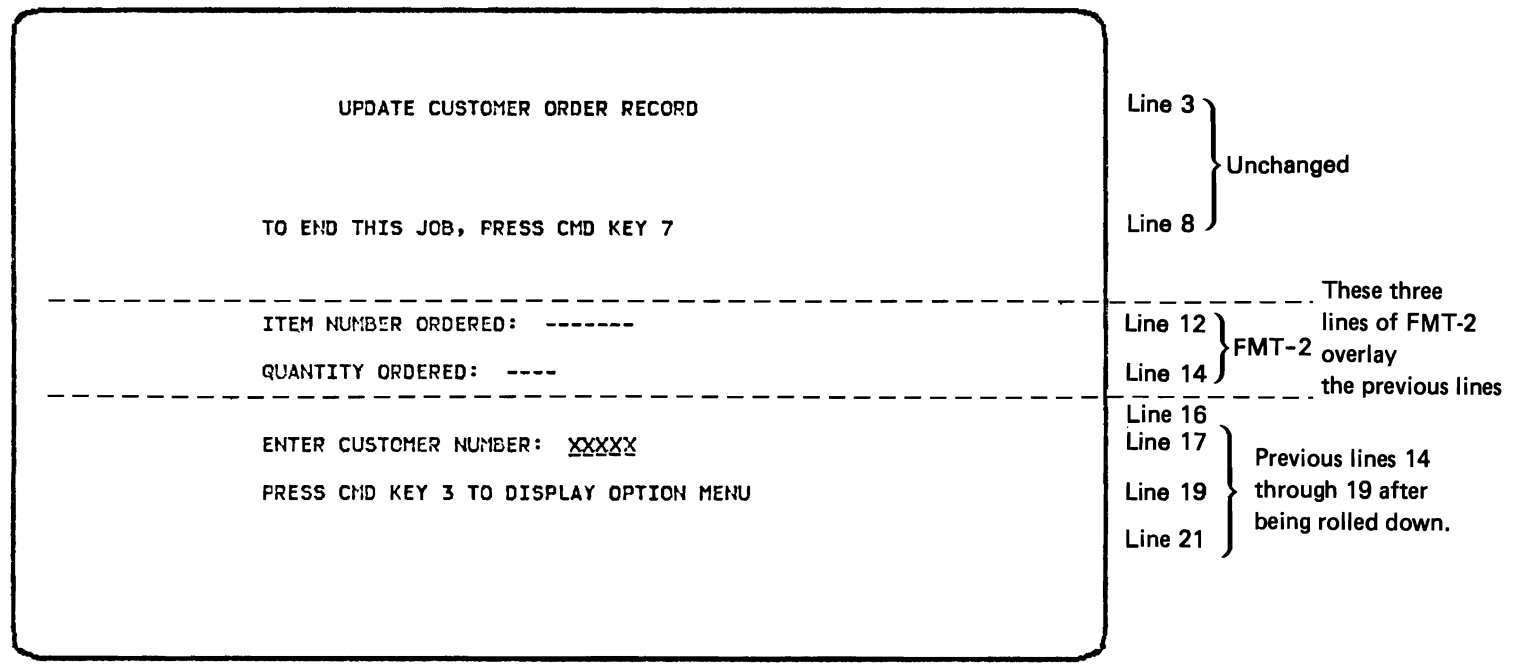


Figure 7-10. Example of ROLLING Operation

SAMPLE COBOL TRANSACTION FILE PROGRAM (MRTSAM)

MRTSAM, shown in Figure 7-13, supports multiple requesting terminals. It sets up two individual work areas and can therefore support two requestors. Normally, this program should be compiled with a MRTMAX value of two, which allows only two display stations to execute this program concurrently.

This program uses two SFGR screen formats, S1 and S2. (Figure 7-11 shows how the screens are displayed; Figure 7-12 shows a partial listing of the SFGR compilation. S1 is a screen that requests the operator to enter a room number. The program uses the room number as the relative record number to access a disk file whose records contain information about the room (guest information, rate information, and so on). S2 is a reply screen that displays the information about the room.

MRTSAM Program Logic

When the program is initiated, it must first open the files and clear the work areas. Then a READ must be executed, which on the first cycle through the program is a read of a blank record from the display station (a status key value of 01 is written into the Function Key area). When the TRANSACTION file READ is satisfied, the display station is allocated a work area identified by the subscript T. When the display station is released from the program, its work area is cleared to blanks.

The AT END option is executed when all display stations have been released from the program. The program recognizes that no display stations are attached when a READ operation is attempted and there are no display stations attached that are capable of sending data (in other words, there are no display stations for which there exists an outstanding invite). This test should be included in all MRT programs.

There are two ways to release a display station from this program. If the operator presses command key 7, the program releases the display station and its work area is cleared. If the display station is released by the operator using the INQUIRY/CANCEL option, a return code of 9D is written in the FILE STATUS. This program tests for this value, and clears the work area for the display station that was released if a 9D is present. The 88 conditional RELEASED-BY-INQUIRY is used for this test.

The indicators in this program are defined as an array with the first field equated to indicator number 1. Therefore, a 10 represents indicator 10. In this program, indicator 10 causes the field ROOMRATE to blink (indicator 10 is set on if the rate for the room is greater than $20.00). Indicator 6 causes the GUESTNAME field to blink on the display screen (indicator 6 is set on when GUESTNAME read from disk is blank).

```
                    GUEST INQUIRY BY ROOM


                    1 - ENTER ROOM #  001




                    2 - PRESS ENTER
                            OR
                    CMD KEY 7 TO END JOB.
```

```
                    GUEST INQUIRY BY ROOM


               ROOM # - 002
               RATE   -    $.00
               GUEST  - *** AVAILABLE ***
               STREET -
               CITY   -
               STATE  -

                    1 - ENTER ROOM #

                    2 - PRESS ENTER
                            OR
                    CMD KEY 7 TO END JOB.
```

Figure 7-11. Display Screens for Sample Program (MRTSAM)

```
  COBFMTS - SOURCE MEMBER NAME

00010SS1
     D*  1         2         3         4         5         6         7         8
     D*  01234567890123456789012345678901234567890123456789012345678901234567890
00020DSCREEN     10102Y  Y           Y     Y           C1
00030DFL01      210526Y                 Y       Y      CGUEST INQUIRY BY ROOM
00040DFL02      170926Y                                C1 - ENTER ROOM #
00060DROOMNO  00030944   YN Y ZY  YY  Y     Y   Y
00080DFA0002  00201726Y                                C2 - PRESS FIELD EXIT
00070DFL04       21934Y                                COR
00080DFL05      232126Y                                CCMD KEY 7 TO END JOB.
     D*  1         2         3         4         5         6         7         8
     D*  01234567890123456789012345678901234567890123456789012345678901234567890


  INPUT BUFFER DESCRIPTION

  FIELD                          START          END
  NAME            LENGTH         POSITION       POSITION

  SCREEN             1              1              1
  ROOMNO             3              2              4
```

Figure 7-12 (Part 1 of 2). Display Screen Format Specifications Listing for Sample Program (MRTSAM)

```
  COBFMTS - SOURCE MEMBER NAME

00090SS2              N
     D*  1         2         3         4         5         6         7         8
     D*  01234567890123456789012345678901234567890123456789012345678901234567890
00100DSCREEN     10102Y  Y           Y     Y              C2
00110DFL01      210528Y                 Y       Y        CGUEST INQUIRY BY ROOM
00120DFL001      70822Y                                   CROOM #-
00250DROOMNUMB00030830Y
00140DFL0002     70922Y                                   CRATE  -
00250DROOMRATE00070930Y                   10
00160DFL0003     71022Y                                   CGUEST -
00250DGUESTNAM00251030Y                   06
00180DFL0004     71122Y                                   CSTREET-
00250DSTREET  00251130Y
00200DFL0005     71222Y                                   CCITY  -
00250DCITY    00151230Y
00220DFL0006     71322Y                                   CSTATE -
00250DSTATE   00021330Y
00240DFL02      171526Y                                   C1 - ENTER ROOM #
00250DROOMNO  00031544   Y  Y ZY  YY  Y     Y   Y
00260DFL03      231726Y                                   C2 - PRESS FIELD EXIT
00270DFL04       21934Y                                   COR
00280DFL05      232126Y                                   CCMD KEY 7 TO END JOB.
     D*  1         2         3         4         5         6         7         8
     D*  01234567890123456789012345678901234567890123456789012345678901234567890


  EXECUTION TIME OUTPUT BUFFER DESCRIPTION

  FIELD                          START          END
  NAME              LENGTH       POSITION       POSITION

  ROOMNUMB             3            1              3
  ROOMRATE             7            4             10
  GUESTNAM            25           11             35
  STREET              25           36             60
  CITY                15           61             75
  STATE                2           76             77


  INPUT BUFFER DESCRIPTION

  FIELD                          START          END
  NAME              LENGTH       POSITION       POSITION

  SCREEN               1            1              1
  ROOMNO               3            2              4


  KRLLIB  - INPUT LIBRARY NAME

  KRLLIB  - OUTPUT LIBRARY NAME

  COBFMTS - FORMAT LOAD MEMBER NAME
```

Figure 7-12 (Part 2 of 2). Display Screen Format Specifications Listing for Sample Program (MRTSAM)

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S .........IDENTFCN SEQ/NO S

     PROCESS LET,NOHALT                                                    000100
   1 IDENTIFICATION DIVISION.                                              000200
   2   PROGRAM-ID. MRTSAMPLE.                                              000300
     *-------------------------------------------------------------------| 000400
   3 SECURITY.                                                           | 000500
         THIS IS A SAMPLE MULTIPLE REQUESTING TERMINAL (MRT) PROGRAM     | 000600
         DESIGNED TO RUN ON A SYSTEM/34. A 3 DIGIT ROOM NUMBER IS        | 000700
         ENTERED FROM A DISPLAY STATION ATTACHED TO THE PROGRAM.         | 000800
                                                                         | 000900
         THE ROOM NUMBER IS READ FROM THE DISPLAY AND IS USED TO         | 001000
         READ A FILE WHOSE RECORDS CONTAIN GUEST NAME AND ROOM RATE      | 001100
         INFORMATION. THE PROGRAM DISPLAYS THE GUEST/ROOM INFORMATION    | 001200
         ON THE REQUESTING DISPLAY. ERRORS ARE DISPLAYED WITH THE        | 001300
         WRITE ERROR FUNCTION. YOUR DISPLAY STATION IS RELEASED BY       | 001400
         USING COMMAND KEY 7. THE PROGRAM ENDS WHEN THERE ARE NO         | 001500
         DISPLAY STATIONS ATTACHED TO THE PROGRAM.                       | 001600
                                                                         | 001700
         TWO SCREENS FORMATS ARE USED (S1, S2) THAT WERE GENERATED       | 001800
         SCREEN FORMAT GENERATOR ROUTINE (SFGR). THEY ARE STORED IN      | 001900
         FORMAT LOAD MEMBER 'COBFMTS'                                    | 002000
     *-------------------------------------------------------------------| 002100
                                                                           002200
   4 ENVIRONMENT DIVISION.                                                 002300
   5   CONFIGURATION SECTION.                                              002400
   6    SOURCE-COMPUTER. IBM-S34.                                          002500
     *-------------------------------------------------------------------| 002600
     *     REMOVE * IN COL 7 OF FOLLOWING STATEMENT TO COMPILE DEBUG     | 002700
     *         STATEMENTS INTO PROGRAM                                   | 002800
     *-------------------------------------------------------------------| 002900
     *      WITH DEBUGGING MODE.                                           003000
(A)  7    OBJECT-COMPUTER. IBM-S34.                                       003100
                                                                           003200
   8   INPUT-OUTPUT SECTION.                                               003300
   9    FILE-CONTROL.                                                      003400
                                                                           003500
  10     SELECT GUEST-FILE ASSIGN TO DISK-GSTFILE                         003600
             ORGANIZATION IS RELATIVE                                      003700
             ACCESS IS RANDOM, RELATIVE KEY IS GUEST-KEY.                  003800
                                                                           003900
[1] 11     SELECT SCREEN-FILE ASSIGN TO WORKSTATION-COBFMTS-02,             004000
             ORGANIZATION IS TRANSACTION,                                  004100
             FILE STATUS IS DS-RETURN-CODE,                                004200
             CONTROL-AREA IS DS-CONTROL-AREA.                              004300
                                                                           004400
  12 DATA DIVISION.                                                        004500
  13   FILE SECTION.                                                       004600
                                                                           004700
[2] 14   FD  SCREEN-FILE LABEL RECORDS ARE OMITTED.                        004800
  15   01  SCREEN-RECORD.                                                  004900
  16     05  SCREEN-CODE                  PIC S9.                         005000
  17         88 S1-SCREEN                          VALUE 1.               005100
  18         88 S2-SCREEN                          VALUE 2.               005200
  19     05  ROOMNO                       PIC S999.                       005300
  20     05  FILLER                       PIC X(76).                      005400
                                                                           005500
  21   FD  GUEST-FILE LABEL RECORDS ARE STANDARD.                          005600
  22   01  GUEST-REC.                                                      005700
  23     05  ROOM-NUMBER                  PIC S999.                       005800
  24     05  ROOM-RATE                    PIC S999V99.                    005900
  25     05  GUEST-NAME                   PIC X(25).                      006000
  26     05  STREET-ADD                   PIC X(25).                      006100
  27     05  CITY                         PIC X(15).                      006200
  28     05  STATE                        PIC XX.                         006300
  29     05  FILLER                       PIC X(5).                       006400
```

Figure 7-13 (Part 1 of 5). TRANSACTION File Sample Program (MRTSAM)

```
                                                                        006500
30   WORKING-STORAGE SECTION.                                           006600
                                                                        006700
31   01  ERROR-MESSAGES.                                                006800
32       03  ERROR-MESSAGE                PIC X(78) VALUE SPACES.      006900
33       03  ROOM-HELD-ERROR-MSG          PIC X(28)  VALUE             007000
             'ROOM HELD -- TRY AGAIN LATER'.                            007100
34       03  INVALID-ROOM-NUMBER-MSG      PIC X(32)  VALUE             007200
               'ROOM NUMBER MUST BE FROM 1 TO 10'.                      007300
35       03  INVALID-KEY-MSG              PIC X(54)  VALUE             007400
             'ONLY CMD KEY 7 FOR END OF JOB AND ENTER KEYS ARE VALID'.  007500
36       03  MAX-DISPLAYS-ERROR-MSG       PIC X(54)  VALUE             007600
             'MAXIMUM DISPLAY STATIONS SIGNED-ON TO PROGRAM -- RETRY'.  007700
                                                                        007800
     *-----------------------------------------------------------------|007900
     *      TO CHANGE THE NUMBER OF REQUESTORS HANDLED BY PROGRAM      |008000
     *       CHANGE - MAX-DISPLAYS-ALLOWED - TO NEW MAXIMUM NUMBER     |008100
     *       CHANGE - SPACE-IN-TABLE - TO MAXIMUM NUMBER LESS 1        |008200
     *       CHANGE - MAX-REQUESTORS-ALLOWED - TO NEW MAXIMUM NUMBER   |008300
     *       MRTMAX SHOULD BE THE SAME AS MAX-REQUESTORS-ALLOWED       |008400
     *       MRTMAX IS SPECIFIED AS A COMPILE OPTION OF 'COBOL' PROC   |008500
     *-----------------------------------------------------------------|008600
                                                                        008700
37     01  PROGRAM-VARIABLES.                                          008800
38         03  NUMBER-OF-SESSIONS         PIC S99.                     008900
39             88 MAX-DISPLAYS-ALLOWED               VALUE 2.         009000
40             88 SPACE-IN-TABLE                     VALUES 0 THRU 1. 009100
41         03  MAX-REQUESTORS-ALLOWED     PIC S99   VALUE 2.         009200
42         03  GUEST-KEY                  PIC S999  VALUE ZERO.      009300
43         03  VALID-ROOM-FLAG            PIC X(3).                    009400
44             88 VALID-ROOM                         VALUE 'YES'.     009500
3  45      03  DS-RETURN-CODE             PIC XX.                      009600
46             88 END-OF-FILE                        VALUE '10'.      009700
47             88 NEW-REQUESTOR                      VALUE '01'.      009800
48             88 SUCESSFUL-READ                     VALUE '00'.      009900
49             88 RELEASED-BY-INQUIRY                VALUE '9D'.      010000
                                                                        010100
4  50   01  DS-CONTROL-AREA.                                           010200
51          03  DS-AID-BYTE               PIC S99.                     010300
52              88 ENTER-KEY                          VALUE 0.         010400
53              88 CMD-KEY7                           VALUE 7.         010500
54          03  SESSION-ID                PIC XX.                      010600
55          03  FILLER                    PIC X(08).                   010700
                                                                        010800
     *-----------------------------------------------------------------|010900
     *     HOLD AREAS FOR EACH DISPLAY STATION                        |011000
     *-----------------------------------------------------------------|011100
56   01  SAVE-AREAS.                                                   011200
5  57      02 HOLD-AREA OCCURS 1 TO 16 TIMES                           011300
               DEPENDING ON MAX-REQUESTORS-ALLOWED                      011400
               INDEXED BY T, T2.                                        011500
58           03  SAVE-ID                  PIC XX.                      011600
59           03  SAVE-ROOM                PIC XXX.                     011700
                                                                        011800
     *-----------------------------------------------------------------|011900
     *            SCREEN S2 OUTPUT AREA                               |012000
     *-----------------------------------------------------------------|012100
6  60   01  S2-OUTPUT-AREA.                                            012200
61        05  ROOM-NUMBER                 PIC XXX.                     012300
62        05  ROOM-RATE                   PIC $$$$.99.                 012400
63        05  GUEST-NAME                  PIC X(25).                   012500
64        05  STREET-ADD                  PIC X(25).                   012600
65        05  CITY                        PIC X(15).                   012700
66        05  STATE                       PIC XX.                      012800
67        05  FILLER                      PIC XXX.                     012900
```

Figure 7-13 (Part 2 of 5). TRANSACTION File Sample Program (MRTSAM)

```
                                                                        013000
     *-------------------------------------------------------------|      013100
     *SCREEN FORMAT INDICATOR TABLE. MAY BE UP TO 99 DIGITS IN SIZE.  *      013200
     *-------------------------------------------------------------|      013300
  68  01  SCREEN-INDICATORS.                                                013400
  69      05  IND                          PIC  1 OCCURS 10 TIMES          013500
                                                INDICATOR 1.              013600
                                                                          013700
                                                                          013800
  70  PROCEDURE DIVISION.                                                 013900
                                                                          014000
                                                                          014100
                                                                          014200
                                                                          014300
  71  DECLARATIVES.                                                       014400
 [7]  72    DEBUG SECTION.                                                014500
           USE FOR DEBUGGING ON ALL PROCEDURES.                           014600
           DISPLAY DEBUG-ITEM.                                            014700
  73  END DECLARATIVES.                                                   014800
                                                                          014900
                                                                          015000
 [8]  74  MAIN-PROCESSING SECTION.                                        015100
  75    MAINLINE.                                                         015200
  76      OPEN INPUT GUEST-FILE                                           015300
               I-O SCREEN-FILE.                                           015400
  77      MOVE SPACES TO SAVE-AREAS, S2-OUTPUT-AREA.                      015500
  78      PERFORM PROCESS-TRANSACTIONS                                    015600
              UNTIL END-OF-FILE.                                          015700
  79      CLOSE GUEST-FILE                                                015800
                SCREEN-FILE.                                              015900
  80      STOP RUN.                                                       016000
                                                                          016100
     *-------------------------------------------------------------|      016300
 [9] *    THE FOLLOWING PROCEDURES ARE EXECUTED BY PERFORM STATEMENTS |     016400
     *         THERE IS NO FALL THROUGH LOGIC                        |     016500
     *-------------------------------------------------------------|      016600
                                                                          016700
  81  PROCESS-TRANSACTIONS.                                               016800
  82      MOVE 'NO' TO VALID-ROOM-FLAG.                                   016900
  83      PERFORM GET-ROOM                                                017000
              UNTIL VALID-ROOM OR END-OF-FILE.                            017100
  84      IF VALID-ROOM                                                   017200
  85          PERFORM RETRIEVE-AND-DISPLAY-ROOM-REC.                      017300
                                                                          017400
                                                                          017500
[10]  86  GET-ROOM.                                                       017600
  87      READ SCREEN-FILE,                                               017700
  88          AT END SET END-OF-FILE TO TRUE.                             017800
        D     PERFORM DISPLAY-READ-DATA.                                  017900
  89      IF RELEASED-BY-INQUIRY OR SUCESSFUL-READ                        018000
  90          PERFORM FIND-DISPLAY-ID                                     018100
  91          IF SUCESSFUL-READ AND ENTER-KEY                             018200
  92              PERFORM VALID-ROOM-TEST                                 018300
  93          ELSE                                                        018400
  94              IF CMD-KEY7                                             018500
  95                  PERFORM RELEASE-REQUEST                             018600
  96                  PERFORM BLANK-HOLD-AREA                             018700
  97              ELSE                                                    018800
  98                  IF RELEASED-BY-INQUIRY                              018900
  99                      PERFORM BLANK-HOLD-AREA                         019000
 100                  ELSE                                                019100
 101                      MOVE INVALID-KEY-MSG TO ERROR-MESSAGE           019200
 102                      PERFORM WRITE-ERROR-MESSAGE                     019300
 103      ELSE                                                            019400
 104          IF NEW-REQUESTOR AND SPACE-IN-TABLE                         019500
 105              PERFORM ADD-NEW-REQUESTOR                               019600
 106          ELSE                                                        019700
 107              IF MAX-DISPLAYS-ALLOWED                                 019800
 108                  MOVE MAX-DISPLAYS-ERROR-MSG TO ERROR-MESSAGE        019900
 109                  PERFORM WRITE-ERROR-MESSAGE                         020000
 110                  DROP SESSION-ID FROM SCREEN-FILE.                   020100
```

Figure 7-13 (Part 3 of 5). TRANSACTION File Sample Program (MRTSAM)

```
      /                                                                  020200
  111 VALID-ROOM-TEST.                                                   020300
  112     IF ROOMNO IS NOT GREATER THAN 10 AND NOT LESS THAN 1           020400
  113         SET VALID-ROOM TO TRUE                                     020500
  114         PERFORM ROOM-IN-USE-TEST                                   020600
  115     ELSE                                                           020700
  116         MOVE INVALID-ROOM-NUMBER-MSG TO ERROR-MESSAGE              020800
  117         PERFORM WRITE-ERROR-MESSAGE.                               020900
                                                                         021000
                                                                         021100
[11] 118 ROOM-IN-USE-TEST.                                               021200
  119     MOVE SPACES TO SAVE-ROOM (T)                                   021300
  120     SET T2 TO 1                                                    021400
  121     SEARCH HOLD-AREA VARYING T2                                    021500
  122         WHEN SAVE-ROOM (T2) IS EQUAL TO ROOMNO                     021600
  123             MOVE ROOM-HELD-ERROR-MSG TO ERROR-MESSAGE              021700
  124             PERFORM WRITE-ERROR-MESSAGE                            021800
  125             MOVE 'NO' TO VALID-ROOM-FLAG.                          021900
                                                                         022000
                                                                         022100
  126 FIND-DISPLAY-ID.                                                   022200
  127     SET T TO 1                                                     022300
  128     SEARCH HOLD-AREA VARYING T                                     022400
  129         WHEN SAVE-ID (T) IS EQUAL TO SESSION-ID NEXT SENTENCE.     022500
                                                                         022600
                                                                         022700
  130 RELEASE-REQUEST.                                                   022800
  131     DROP SESSION-ID FROM SCREEN-FILE.                              022900
                                                                         023000
                                                                         023100
  132 BLANK-HOLD-AREA.                                                   023200
  133     MOVE SPACES TO SAVE-ID (T), SAVE-ROOM (T)                      023300
  134     SUBTRACT 1 FROM NUMBER-OF-SESSIONS.                            023400
                                                                         023500
  135 ADD-NEW-REQUESTOR.                                                 023600
  136     PERFORM FIND-BLANK-ENTRY                                       023700
  137     MOVE SESSION-ID TO SAVE-ID (T).                                023800
  138     MOVE ZEROS TO SCREEN-INDICATORS, ROOMNO.                       023900
  139     ADD 1 TO NUMBER-OF-SESSIONS.                                   024000
  140     WRITE SCREEN-RECORD FORMAT IS 'S1'.                            024100
                                                                         024200
  141 FIND-BLANK-ENTRY.                                                  024300
  142     SET T TO 1                                                     024400
  143     SEARCH HOLD-AREA VARYING T                                     024500
  144         WHEN SAVE-ID (T) EQUAL SPACES NEXT SENTENCE.               024600
                                                                         024700
                                                                         024800
  145 RETRIEVE-AND-DISPLAY-ROOM-REC.                                     024900
  146     MOVE ROOMNO TO GUEST-KEY.                                      025000
  147     READ GUEST-FILE, INVALID KEY                                   025100
  148         MOVE INVALID-ROOM-NUMBER-MSG TO ERROR-MESSAGE              025200
  149         PERFORM WRITE-ERROR-MESSAGE                                025300
  150             MOVE 'NO' TO VALID-ROOM-FLAG.                          025400
  151     IF VALID-ROOM                                                  025500
  152         PERFORM RATE-AND-AVAILABILITY-TEST                         025600
  153     MOVE ROOMNO TO SAVE-ROOM (T)                                   025700
  154     MOVE CORRESPONDING GUEST-REC TO S2-OUTPUT-AREA                 025800
  155     WRITE SCREEN-RECORD FROM S2-OUTPUT-AREA                        025900
              FORMAT 'S2'                                                026000
              INDICATORS ARE SCREEN-INDICATORS.                          026100
  156     MOVE ZEROS TO SCREEN-INDICATORS.                               026200
                                                                         026300
                                                                         026400
```

Figure 7-13 (Part 4 of 5). TRANSACTION File Sample Program (MRTSAM)

```
157  RATE-AND-AVAILABILITY-TEST.                                          026500
158           IF ROOM-RATE OF GUEST-REC GREATER THAN 20.00                026600
159               MOVE B'1' TO IND (10).                                  026700
160           IF GUEST-NAME OF GUEST-REC EQUAL SPACES                     026800
161               MOVE B'1' TO IND (6)                                    026900
162               MOVE '*** AVAILABLE ***' TO GUEST-NAME OF GUEST-REC.    027000
                                                                          027100
                                                                          027200
163  WRITE-ERROR-MESSAGE.                                                 027300
B 164      WRITE SCREEN-RECORD FROM ERROR-MESSAGE                         027400
               FORMAT 'ERRLINE'.                                          027500
                                                                          027600
                                                                          027700
12       DDISPLAY-READ-DATA.                                              027800
         D    DISPLAY 'KEY USED CODE = ', DS-AID-BYTE,                    027900
         D         '  DISPLAY STATION ID = ', SESSION-ID,                028000
         D         '  RETURN CODE = ', DS-RETURN-CODE,                   028100
         D         '  ROOM NUMBER = ', ROOMNO, '  READ OPERATION'.       028200

PROGRAM SIZE = DATA DIVISION + PROCEDURE DIVISION + LITERALS + DTF/BUFFERS

     3032             616              1455             121          840


NO ERRORS DETECTED FOR THIS COMPILATION

  END OF COMPILATION
```

```
13 SYS-3130 I MRTSAM MODULE'S MAIN STORAGE SIZE IS

          10433 DECIMAL
SYS-3131 I 0000 IS THE START CONTROL ADDRESS OF THIS MODULE
SYS-3134 I MRTSAM MODULE IS CATALOGED AS A LOAD MEMBER
           KRLLIB    IS THE LIBRARY NAME
               46 TOTAL NUMBER OF LIBRARY SECTORS
```

Figure 7-13 (Part 5 of 5). TRANSACTION File Sample Program (MRTSAM)

The following items describe portions of the sample program. The numbers correspond to the numbers on the listing.

(A) To enable debugging mode, do not make comments on this line.

1 The FILE-CONTROL entry specifies a file with organization of TRANSACTION. It is associated with an SFGR load member named COBFMTS which can contain two formats.

2 The FD entry further defines the TRANSACTION file. The size of the record is implicitly defined as 80 bytes. The record area is used for both input and output. All I-O TRANSACTION files must be opened as I-O.

3 DS-RETURN-CODE is the data-name specified by the FILE STATUS clause in the FILE-CONTROL entry. This area is the 2-byte file status return code data area. The file status values for successful read, end-of-file, initial read of a requestor, and requestor released are tested for with 88 level condition names. See Appendix H for a listing of status values.

4 DS-CONTROL-AREA is the data-name defined by the control entry for the TRANSACTION file. The ID of the display station associated with each TRANSACTION file input or output operation is returned to the data area SESSION-ID when the operation is completed. Upon completion of each READ operation, the function key used by the display station operator has its corresponding return code placed in the 2-byte field, DS-AID-BYTE. The remaining 8 bytes, FILLER, are required for compatibility with IBM System/38 COBOL.

5 The program is written to allocate up to 16 individual data areas to hold data unique by attached display station. Only two data areas are allocated here, because MAX-REQUESTORS-ALLOWED was set to 2.

You must determine when unique areas are required. You need unique areas when information about a screen or transaction must be recalled to process the next screen or transaction from a given requestor (for example, batch totals by requestor).

6 This is the screen output record area.

7 An example of debugging code has been included in this program. When debugging is active, the DEBUG SECTION causes the name of every paragraph executed to be displayed. The statement just prior to statement 7 causes debugging (DEBUG SECTION and any statement with a D in column 7) to be compiled into the program when the * in column 7 is removed.

See Figure 7-15 for an example of debugging output.

8 This is the program mainline. It causes all of the lower level procedures to be executed until end-of-file occurs.

9 One of the rules of structured programming is that all procedures are executed by PERFORM statements. There is no fall-through logic.

10 GET-ROOM provides return code (DS-RETURN-CODE) and entry key (DS-AID-BYTE) logic processing. This is the heart of MRT logic processing. The debug line (D in column 7) provides information about the READ when debugging is active. The logic of this procedure could be simplified if SFGR were used to mask out invalid command keys so that it would not be necessary to test for an invalid key in the program. The SPACE-IN-TABLE test (statement 104) would not be necessary if the MRTMAX parameter specified on the compile procedure always had the same value as the maximum number of requestors the program can handle.

For a flowchart of GET-ROOM, see Figure 7-14.

11 This paragraph determines if two different display stations are making an inquiry of the same room.

(B) Refer to "WRITE Statement" in this chapter: "Writing to the Error Line" under "Format Option."

12 Debugging code explained in 7. Debugging was not active when this compiler listing was generated.

13 The additional storage requirement to execute this program with debugging active is approximately 30%.

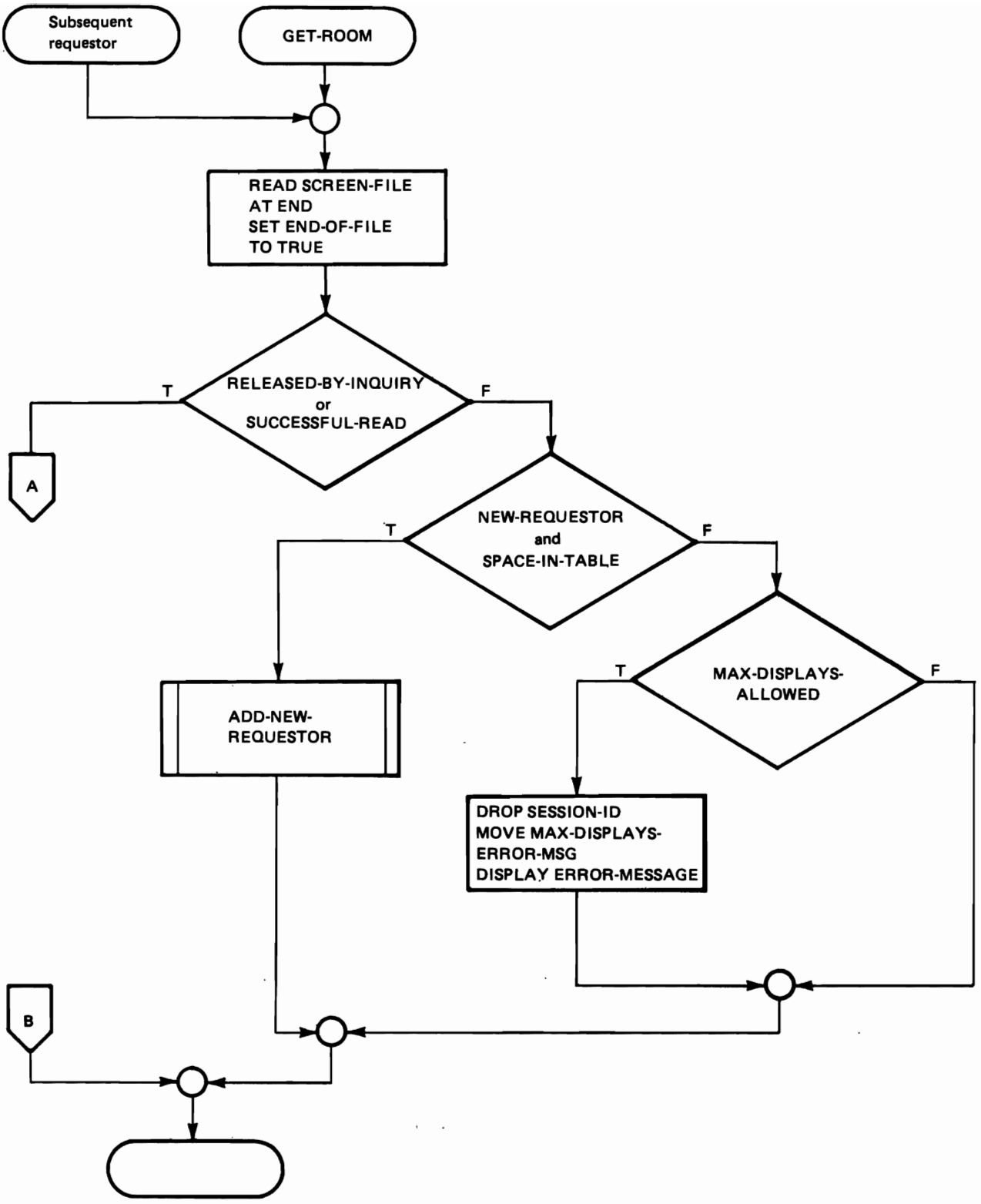


Figure 7-14 (Part 1 of 2). Flowchart of GET-ROOM

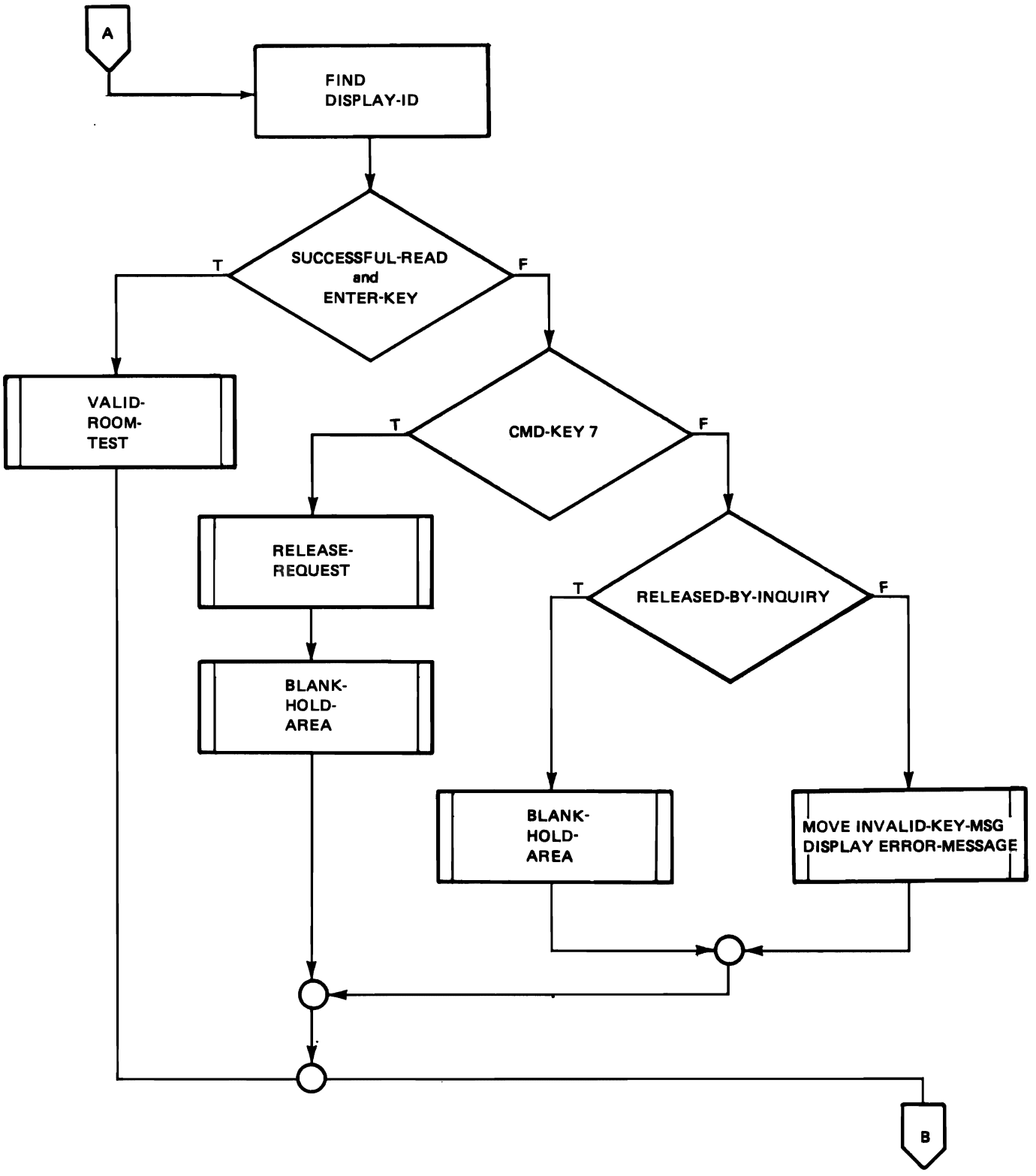


Figure 7-14 (Part 2 of 2). Flowchart of GET-ROOM

This page is intentionally left blank.

MRTSAM Debugging

Debugging code (both the DEBUG SECTION and some additional debugging statements) has been included in MRTSAM. MRT program logic can be very difficult to follow, because different requestors can be at different points in your program. By including debugging code in your MRT program, you can often save yourself time when you are trying to find what caused an error.

The DEBUG SECTION causes the name of every paragraph that is executed to be displayed. This provides you with a record of how your program executed. MRTSAM includes additional debugging statements that display the function key used to enter data, the display station ID, the return code generated by the READ statement to the disk file, and the room number used for this request. This information allows you to narrow down which display station had an input/output error.

Figure 7-15 shows an example of the debugging output for MRTSAM.

```
A 000078 MAIN-PROCESSING                                    START PROGRAM
  000078 MAINLINE                                           FALL THROUGH
  000080 PROCESS-TRANSACTIONS                               PERFORM LOOP
  000085 GET-ROOM                                           PERFORM LOOP
  000091 DISPLAY-READ-DATA                                  PERFORM LOOP
  KEY USED CODE = 00  DISPLAY STATION ID = W4  RETURN CODE = 01  ROOM NUMBER =      READ OPERATION
  000108 ADD-NEW-REQUESTOR                                  PERFORM LOOP
  000139 FIND-BLANK-ENTRY                                   PERFORM LOOP
  000085 GET-ROOM                                           PERFORM LOOP
B 000091 DISPLAY-READ-DATA                                  PERFORM LOOP
  KEY USED CODE = 00  DISPLAY STATION ID = W4  RETURN CODE = 00  ROOM NUMBER = 001  READ OPERATION
  000093 FIND-DISPLAY-ID                                    PERFORM LOOP
  000095 VALID-ROOM-TEST                                    PERFORM LOOP
  000117 ROOM-IN-USE-TEST                                   PERFORM LOOP
  000087 RETRIEVE-AND-DISPLAY-ROOM-REC                      PERFORM LOOP
  000155 RATE-AND-AVAILABILITY-TEST                         PERFORM LOOP
  000080 PROCESS-TRANSACTIONS                               PERFORM LOOP
  000085 GET-ROOM                                           PERFORM LOOP
C 000091 DISPLAY-READ-DATA                                  PERFORM LOOP
  KEY USED CODE = 00  DISPLAY STATION ID = W4  RETURN CODE = 00  ROOM NUMBER = 011  READ OPERATION
  000093 FIND-DISPLAY-ID                                    PERFORM LOOP
  000095 VALID-ROOM-TEST                                    PERFORM LOOP
  000120 WRITE-ERROR-MESSAGE                                PERFORM LOOP
  000085 GET-ROOM                                           PERFORM LOOP
D 000091 DISPLAY-READ-DATA                                  PERFORM LOOP
  KEY USED CODE = 00  DISPLAY STATION ID = W1  RETURN CODE = 01  ROOM NUMBER = 00M  READ OPERATION
  000108 ADD-NEW-REQUESTOR                                  PERFORM LOOP
  000139 FIND-BLANK-ENTRY                                   PERFORM LOOP
  000085 GET-ROOM                                           PERFORM LOOP
E 000091 DISPLAY-READ-DATA                                  PERFORM LOOP
  KEY USED CODE = 00  DISPLAY STATION ID = W2  RETURN CODE = 01  ROOM NUMBER = 000  READ OPERATION
  000112 WRITE-ERROR-MESSAGE                                PERFORM LOOP
  000085 GET-ROOM                                           PERFORM LOOP
F 000091 DISPLAY-READ-DATA                                  PERFORM LOOP
  KEY USED CODE = 00  DISPLAY STATION ID = W1  RETURN CODE = 00  ROOM NUMBER = 004  READ OPERATION
  000093 FIND-DISPLAY-ID                                    PERFORM LOOP
  000095 VALID-ROOM-TEST                                    PERFORM LOOP
  000117 ROOM-IN-USE-TEST                                   PERFORM LOOP
  000087 RETRIEVE-AND-DISPLAY-ROOM-REC                      PERFORM LOOP
  000155 RATE-AND-AVAILABILITY-TEST                         PERFORM LOOP
  000080 PROCESS-TRANSACTIONS                               PERFORM LOOP
  000085 GET-ROOM                                           PERFORM LOOP
G 000091 DISPLAY-READ-DATA                                  PERFORM LOOP
  KEY USED CODE = 00  DISPLAY STATION ID = W1  RETURN CODE = 00  ROOM NUMBER = 001  READ OPERATION
  000093 FIND-DISPLAY-ID                                    PERFORM LOOP
  000095 VALID-ROOM-TEST                                    PERFORM LOOP
  000117 ROOM-IN-USE-TEST                                   PERFORM LOOP
  000127 WRITE-ERROR-MESSAGE                                PERFORM LOOP
  000085 GET-ROOM                                           PERFORM LOOP
H 000091 DISPLAY-READ-DATA                                  PERFORM LOOP
  KEY USED CODE = 02  DISPLAY STATION ID = W1  RETURN CODE = 00  ROOM NUMBER = 001  READ OPERATION
  000093 FIND-DISPLAY-ID                                    PERFORM LOOP
  000105 WRITE-ERROR-MESSAGE                                PERFORM LOOP
  000085 GET-ROOM                                           PERFORM LOOP
I 000091 DISPLAY-READ-DATA                                  PERFORM LOOP
  KEY USED CODE = 99  DISPLAY STATION ID = W1  RETURN CODE = 9D  ROOM NUMBER = NLY  READ OPERATION
  000093 FIND-DISPLAY-ID                                    PERFORM LOOP
  000102 BLANK-HOLD-AREA                                    PERFORM LOOP
  000085 GET-ROOM                                           PERFORM LOOP
J 000091 DISPLAY-READ-DATA                                  PERFORM LOOP
  KEY USED CODE = 07  DISPLAY STATION ID = W4  RETURN CODE = 00  ROOM NUMBER = 011  READ OPERATION
  000093 FIND-DISPLAY-ID                                    PERFORM LOOP
  000098 RELEASE-REQUEST                                    PERFORM LOOP
  000099 BLANK-HOLD-AREA                                    PERFORM LOOP
  000085 GET-ROOM                                           PERFORM LOOP
K 000091 DISPLAY-READ-DATA                                  PERFORM LOOP
  KEY USED CODE = 00  DISPLAY STATION ID =     RETURN CODE = 10  ROOM NUMBER = 011  READ OPERATION
```

Figure 7-15. Debugging output for MRTSAM

MRTSAM Debug Output, Sequence of Events

Ⓐ Display station W4 signs on.

Ⓑ Display station W4 enters room number 001.

Ⓒ Display station W4 enters room number 011. INVALID-ROOM-NUMBER-MSG is displayed using WRITE to ERRLINE.

Ⓓ Display station W1 signs on.

Ⓔ Display station W2 signs on. MAX-DISPLAYS-ERROR-MSG is displayed. The program is written to handle only two requestors. The program was compiled to handle three requestors. The difference between the number of requestors the program can handle and the number of requestors specified on the compile is a common error. This program contains a routine that checks for this error condition.

Ⓕ Display station W1 enters room number 004.

Ⓖ Display station W1 enters room number 001. ROOM-HELD-ERROR-MSG is displayed.

Ⓗ Display station W1 tries to use command key 2 to release itself from the program. INVALID-KEY-MSG is displayed.

Ⓘ Display station W1 uses INQUIRY and a 3 response to release itself from the program.

Ⓙ Display station W4 uses command key 7 to release itself from the program.

Ⓚ End-of-file occurs on the READ.

Chapter 8. System-Dependent Considerations

This chapter documents the various system-defined limits and flexibilities that apply to System/34 COBOL. These items should be considered when writing a COBOL program for System/34.

GENERAL CONSIDERATIONS

Library-Name, Program-Name, and Text-Name

Unique entries must be specified for library-name, program-name, and COPY text-name. Although these names can be up to 30 characters in length, they must meet the following restrictions:

- Library-name must be unique within the first 8 characters.
- Program-name must be unique within the first 6 characters.
- COPY text-name must be unique within the first 8 characters.

The remaining characters of each of these names are used only for documentation.

Source Statements

The maximum number of source statements is 65535. A source statement is defined as:

Procedure Division–a COBOL statement
Other Divisions–a source line

Source Program Library

If the OF/IN option of the COPY statement is not specified; the library defaults to the LIBRARY option of the PROCESS statement. If the LIBRARY option is not specified, the default is #LIBRARY.

User-Defined Words

No more than 32767 user-defined words are allowed in a program.

Files

A maximum of 25 files (FD or SD) can be defined in a COBOL program.

Disk Data Management

System/34 offers the flexibility of allowing you to define and process indexed and relative files as if they were defined as physical sequential files. You can also define and process sequential and indexed files as if they were defined as relative files. For more information on file organization and access modes, see *File Processing Summary* in Chapter 3.

Indexed and Relative File Contents

Position 1 of indexed or relative files cannot contain hex FF (for NATIVE collating sequence, this corresponds to HIGH-VALUE). Binary fields (COMPUTATIONAL-4) should be avoided in position 1, because they could contain this value.

A record key used by an indexed random READ or indexed START statement cannot contain hex FF in any position. Binary fields should also be avoided. The key for an indexed file cannot exceed 29 characters.

Adding Records to an Indexed File

When SEQUENTIAL ACCESS to indexed files is specified, records added by a user program or an IBM-supplied utility cannot be sequentially retrieved until a keysort has been performed by SSP. A keysort is only performed during program initiation when DISP-OLD is specified on the FILE OCL statement and SEQUENTIAL or DYNAMIC ACCESS is specified.

When DYNAMIC ACCESS is specified and an indexed file is opened in I-O mode, the WRITE operation is not allowed. That is, records cannot be added to the file except when ACCESS is RANDOM.

ENVIRONMENT DIVISION CONSIDERATIONS

APPLY Clause

The APPLY clause can reference a data name no larger than 9999 bytes.

ASSIGN Clause

The ASSIGN clause associates a file with an external medium. The assignment-name has the following format for printer and disk files:

Device Type-Name

Device Type: PRINTER Printer files
DISK Disk files

Name: A 1- to 8-character field specifying the external name by which the file is known to the system. This is the name that appears in the NAME field on the FILE or PRINTER OCL statement.

IBM Extension:

The assignment-name has the following format for TRANSACTION files:

Type [-Name / -Name-Formats]

Type: WORKSTATION

Name: 1- to 8-character name that specifies the external name of the SFGR generated load member that contains the screen formats. This field is not required if the file is to be used with SSP-ICF sessions only.

Formats: A two-digit numeric value that is equal to or greater than the number of formats in the SFGR load member referenced in the name field. The maximum value and the default value for the number of formats is 32. This field is not required if the file is to be used with SSP-ICF sessions only.

RESERVE Clause

This clause must specify a value of 1 or 2; a minimum of one buffer is required for a file. If this clause is omitted, a minimum of one buffer area is reserved. If the user specifies a value greater than 2, the compiler assigns the value 2.

RERUN Clause

The maximum integer value for the integer-1 RECORDS option is 32767.

All files used in the program must be opened before the first checkpoint can be taken.

SAME RECORD AREA Clause

A maximum of 15 SAME RECORD AREA clauses can be defined in a COBOL program.

SAME AREA or SAME SORT-MERGE AREA Clauses

A maximum of 15 SAME AREA clauses can be defined in a COBOL program. A SAME AREA clause consists of any SAME AREA clause, any SAME SORT AREA clause, or any SAME SORT-MERGE clause.

OBJECT-COMPUTER MEMORY SIZE Clause

This clause must specify an integer from 1 to 65536.

KEY Clause

The RELATIVE KEY data-name must have a length of 1 through 7, and RECORD KEY must be 1 through 29 bytes long.

DATA DIVISION CONSIDERATIONS

BLOCK CONTAINS Clause

The maximum block size is 9999 characters.

RECORD CONTAINS Clause

The maximum record length is 4096 bytes.

LINAGE Clause

The maximum size of the logical page is 32767 lines. The logical page size is the sum of the number of lines in the body, top margin, and bottom margin.

OCCURS Clause

The literals in the OCCURS clause must have a value of 1 through 32767.

Item Size

If no other restrictions apply, the maximum item size is 32767.

Index and Subscript Literals

An index or subscript literal must have a value of 1 through 32767.

PROCEDURE DIVISION CONSIDERATIONS

CALL Statement

A maximum of 20 subroutines can be called by a program. Called subroutines can also reside in the overlay area of the executable program. All subroutines that you want to overlay must have a category number greater than 7 when link-edited. For a further description of overlay, refer to *Link Editing with Overlay* under *Program Linkage* in Chapter 9.

Programs containing independent segments must not be called by another program.

USING Option

A maximum of 15 operands can be specified for the USING option. If the called subprogram is written in a language other than COBOL, a CALL statement USING identifier may be a file-name for a physical sequential file or a data-item defined in the File, Working-Storage, or Linkage Section of the calling program. When a file-name is specified, the identified file must not be opened in the calling program.

COMPUTE Statement

The maximum size of each operand is 18 decimal digits. Division by zero always results in a size error condition.

GO TO DEPENDING ON Statement

The maximum number of branch points that can be specified in one GO TO DEPENDING ON statement is 99.

INSPECT Statement

A maximum of 15 comparison operands (TALLYING/REPLACING) can be specified in an INSPECT statement.

SORT/MERGE Statement

No more than 12 KEYS can be specified in any SORT or MERGE statement. No more than 8 input files can be specified in any SORT or MERGE statement.

STOP Statement

When STOP literal is specified and the literal is nonnumeric, the literal is limited to 120 characters.

UNSTRING Statement

A maximum of 15 delimiters can be specified in an UNSTRING statement, and each delimiter must be an alphanumeric data item.

TRANSACTION File

If your display station is attached to a TRANSACTION file and a SYSLIST procedure or OCL statement was executed, which changed the SYSLIST device to CRT, undesireable results can occur when low-volume data is sent to the display station by the COBOL program (through the use of a low-volume output command, such as DISPLAY, EXHIBIT, or ACCEPT).

Chapter 9. Creating, Executing, and Debugging Programs

GENERAL OVERVIEW

A COBOL program is processed by the COBOL compiler under control of the System Support Program Product (SSP). The COBOL compiler is a program that translates COBOL statements into instructions that can be understood and executed by the system. The SSP is a program that controls the operation of the system.

To make the best use of the system, you must know how to tell the SSP about your COBOL program, how to define COBOL files, and what kind of output to expect. This introductory section summarizes basic information you need in order to use the system and briefly describes:

1. How a COBOL program is processed
2. Communicating with the system through command statements or operation control language (OCL)
3. Program output
4. Defining COBOL files and other files needed by the system

How a COBOL Program is Processed

Before your COBOL program can be executed, it must be converted into a form that can be understood by the system. The compiler converts the program. The linkage editor combines the program with whatever other programs are required to form an executable unit.

The three steps that must be taken to convert and execute a COBOL program are compilation, link-editing, and load module execution. The COBOL source program is the input to the compilation step. The output is the group of translated statements, called an object module, which becomes the input to the link-edit step. The link-editing step combines the object module with other modules to form a load module or object program. The load module is the program executed in the load module execution step.

Although these three steps must be taken in sequence to execute a program, it is not necessary that they occur at one time. For example, you can perform the compilation step only, with the other steps to follow at a later time. Assume that you have coded a particularly complex COBOL program. The first time you submit it, you might only compile it, so that you can correct any source program errors. The compiler examines each COBOL statement for correct syntax and issues error messages for COBOL language violations. After correcting these errors you could have the program compiled, link-edited, and executed at one time.

Further assume that you intend to use the program many times. Once the program is successfully compiled, it would be pointless for you to compile it every time you use it. You could choose to store the compiled and link-edited load module in the library. Then each time you want to execute the program, you could tell the system to bypass the compile and link-edit steps and use the load module as its input. These are some of the alternatives you have when executing your program. You tell the system which alternatives to select through use of the operation control language statements or PROCESS statement options.

Using COBOL consists of the general operations illustrated in Figure 9-1:

1. Define the job. The programmer defines the job requirements for the specific task. Usually, the following questions must be answered when the job is defined:
   a. What information is provided as input to the program?
   b. What operations are to be performed?
   c. What output information should be generated by the program?
2. Write the source program. After the programmer defines the job, he develops the COBOL source program.

1. Job requirements:
   a. Input
   b. Processing
   c. Output

2. Develop source code

3. Record source statements on disk or diskette, or enter them directly during compile step.

4. Compile the source statements. The resulting object module is recorded on disk, and a diagnosed source file can be created.

5. Link-edit the object module. The resulting load module is recorded on disk.

6. Execute the program. The load module is read from disk; then the input and output are processed by the system under the COBOL program.

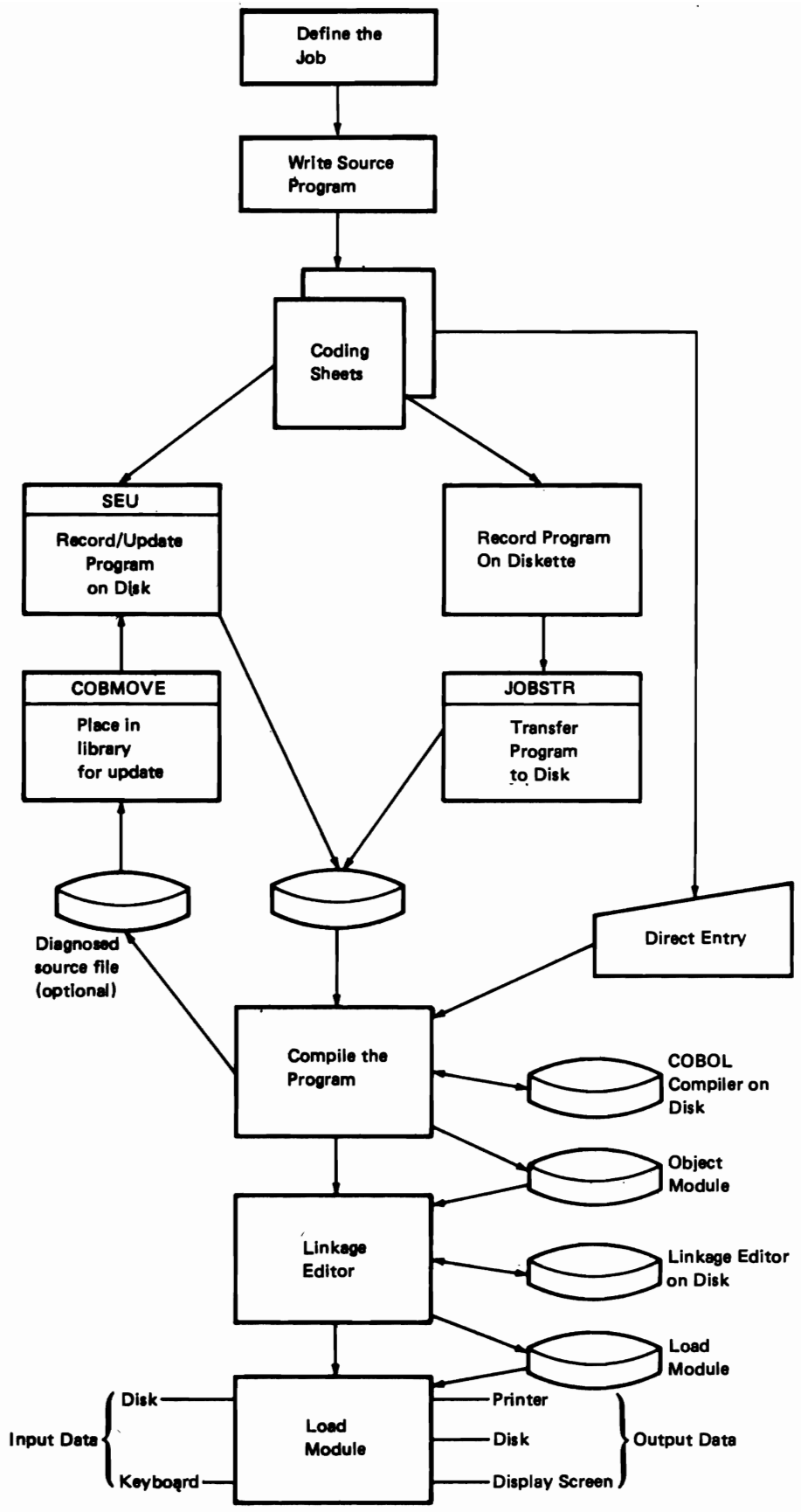


Figure 9-1. Processing a COBOL Program

3. Record the source statements on disk or diskette. Statements can also be entered directly during the compile step. After the source program is written, it is entered on diskette and transferred to disk or entered into the system library from the keyboard, using the Source Entry Utility (see *SEU Reference Manual*) or $MAINT (see *System Support Reference Manual*).

4. Compile the source statements. The source program, preceded by the required OCL statements, is processed by the COBOL compiler under control of the SSP. At the end of this processing (compilation), the object module is stored in the library on disk. This program contains all the instructions required to perform the job, except referenced and implied subroutines. If a diagnosed source file was requested, the COBMOVE procedure can be used to place the file in the library. SEU can then be used to correct compilation errors in the source program.

5. Link-edit the object module. The object module is processed by the linkage editor under control of the system control program. This is done to resolve all addresses and external references. At the end of this processing (link-editing), the load module is stored in the library on disk. The object program is now ready to be executed.

6. Execute the program. The load module is read from disk; then the input and output are processed by the system under control of the COBOL program.

Note: The diagnosed source file discussion and listing are presented later in this chapter.

IBM SYSTEM/34 COBOL-SUPPLIED PROCEDURES

IBM supplies several library procedures for use with System/34 COBOL. When the operator enters an appropriate command statement, the IBM-supplied library procedure is either executed or placed on the input job queue. For information on how to place a job on the input job queue, see the *Operator's Guide*.

Library procedures provide for COBOL compilation and link-editing, execution of COBOL programs, movement of a diagnosed source file to a library, and screen prompts, by using the following command statements:

- COBOL—compile a COBOL program
- COBOLCG—compile and execute a COBOL program
- COBOLG—execute a COBOL program with user-provided procedure for additional OCL statements
- COBSYSIN—compile and link-edit a COBOL program entered from the current SYSIN device
- COBMOVE—move a COBOL diagnosed source file to a library
- COBOLP—provide screen prompts for entering, compiling, executing, and correcting COBOL programs

COBOL Command Statement

To compile and link-edit a COBOL source program, the operator enters the COBOL command. The command statement is:

COBOL pgname,size,inlib,outlib,mrtmax,nep,dsf

where:

pgname is the name of the source program to be compiled (maximum length is 8 characters).

size is the number of blocks allocated for the work files used by the compiler (each block is 2560 bytes). The maximum value of *size* is 9999. If this parameter is not specified, the default is 24 blocks. Generally, one additional block is required for each 10 program source statements over 200.

inlib specifies the name of the library that contains the source program. If this parameter is not specified, the system library (#LIBRARY) is assumed.

outlib specifies the name of the library that will contain the object and load modules if the OBJECT and LINK options are not supplied on the PROCESS statement. If the library is not specified on the PROCESS statement or by using this parameter, the default is the system library.

mrtmax identifies the program being compiled as an MRT program and specifies the maximum number of active requesting display stations that can be attached to the program. The maximum value of *mrtmax* is 256. If *mrtmax* is zero or is omitted, the program is not an MRT program.

nep indicates whether the program to be compiled is a never-ending program. Valid values are YES or NO; the default value is NO.

dsf specifies the file label for the diagnosed source file. If not specified, a diagnosed source file will not be created.

If the COBOL command statement is entered without specifying a source program name, a prompting display (see Figure 9-2, Screen 2) assists you in entering the parameters to compile and link-edit a COBOL program.

COBOLCG Command Statement

To compile, link-edit, and execute a COBOL program with one command, the COBOLCG command is used. The command statement format is:

COBOLCG pgname,size,inlib,outlib,
oclmmbr,userlib,mrtmax,nep,dsf

where:

pgname is the name of the source program to be compiled (maximum length is 8 characters).

size is the number of blocks for the work files used by the compiler. The maximum value of *size* is 9999. If this parameter is not specified, the default is 24 blocks. Generally, one additional block is required for each 10 program source statements over 200.

inlib specifies the name of the library that contains the source program. If this parameter is not specified, the system library (#LIBRARY) is assumed.

outlib specifies the name of the library that will contain the object and load modules if the OBJECT and LINK options are not supplied on the PROCESS statement. If the library is not specified on the PROCESS statement or by using this parameter, the default is the system library.

oclmmbr optionally specifies the name of the procedure containing the FILE OCL statements to be used when executing the designated program.

userlib specifies the library that contains the COBOL load module to be executed and will also be used for the *oclmmbr* search. If the userlib name is not specified, the system first searches the library specified by the outlib parameter, then searches the active library, and finally searches the system library for the load module.

mrtmax identifies the program being compiled as an MRT program and specifies the maximum number of active requesting display stations that can be attached to the program. The maximum value of *mrtmax* is 256. If *mrtmax* is zero or is omitted, the program is not an MRT program.

nep indicates whether the program to be compiled is a never-ending program. Valid values are YES or NO; the default value is NO.

dsf specifies the file label for the diagnosed source file. If not specified, a diagnosed source file will not be created.

If the COBOLCG command statement is entered without specifying a source program name, a prompting display (see Figure 9-2, Screen 4) assists you in entering the parameters to compile, link-edit, and execute a COBOL program.

COBOLG Command Statement

To execute a load module and include specialized OCL statements, the COBOLG command is used. The command statement format is:

COBOLG pgname,oclmmbr,userlib

where:

pgname is the name of the COBOL load module to be executed.

oclmmbr optionally specifies the name of the procedure containing the FILE OCL statements to be used when the designated program is executed.

userlib specifies the library that contains the COBOL load module to be executed and will also be used for the *oclmmbr* search. If the userlib name is not specified, the system first searches the active library, and then searches the system library for the load module.

If the COBOLG command statement is entered without specifying a program name, a prompting display (see Figure 9-2, Screen 3) assists you in entering the parameters to execute a COBOL program.

COBSYSIN Command Statement

To compile and link-edit a COBOL source program that is entered from the current SYSIN device. The command statement format is:

```
COBSYSIN
```

If the COBSYSIN command is entered from the keyboard, the user is allowed to enter COBOL source statements, one at a time. If the COBSYSIN command is contained in a user-provided procedure, records are read from the procedure. If the procedure is exhausted, reading continues from the keyboard.

The last source statement must be followed by a /* termination record.

COBMOVE Command Statement

To move a diagnosed source file to a library, the COBMOVE command is used. The command statement format is:

```
COBMOVE dsf,dsflib,dsfret,dsfdel
```

where:

dsf specifies the file label for the diagnosed source file. This parameter is required.

dsflib specifies the name of the library that is to receive the source member. If this parameter is not specified, the default is the system library.

dsfret specifies the disposition of the source member in the library. The only valid value is REPLACE. If this value is not specified, a displayed message requests replacement if a module of that name exists.

dsfdel specifies the deletion value. If DELETE is specified, the diagnosed source file is deleted.

COBOLP Command Statement

To provide the user with screen prompts for compiling and executing COBOL programs, the COBOLP command is used. The command statement format is:

```
COBOLP
```

The COBOLP command initiates a menu display (see Figure 9-2, Screen 1). The response to this menu determines the next step in this procedure. When applicable, defaults appear on each displayed line.

Response	Action Taken
0	This response terminates the COBOLP procedure.
1	A prompting display (see Figure 9-2, Screen 2) assists you in entering the parameters to compile a COBOL program. Source program name is a required parameter. For a further explanation of these parameters, see the COBOL command statement.
2	A prompting display (see Figure 9-2, Screen 3) assists you in entering the parameters to execute a COBOL program. Program name is a required parameter. For a further explanation of these parameters, see the COBOLG command statement.
3	A prompting display (see Figure 9-2, Screen 4) assists you in entering the parameters to compile and execute a COBOL program. Source program name is a required parameter. For a further explanation of these parameters, see the COBOL, COBOLG, and COBOLCG command statements.
4	A prompting display (see Figure 9-2, Screen 5) assists you in entering the parameters to create or update a COBOL module. Source name or procedure module name is a required parameter.
5	A prompting display (see Figure 9-2, Screen 6) assists you in entering the parameters to move a diagnosed source file to a library. Diagnosed Source File Label is a required parameter.

Completion of the requested step (as designated by Response 1 through 5) causes Screen 1 to be re-displayed. If no further steps are required, a response of 0 terminates the COBOLP procedure.

Screen 1

```
                    COBOLP PROCEDURE

0 - COBOL       Exits From COBOL Processing
1 - COBOL       Compiles a COBOL Program
2 - COSOLG      Executes a COBOL Program
3 - COBOLCG     Compiles and Executes a COBOL Program
4 - SEU         Creates Or Updates a COBOL Module
5 - COBMOVE     Moves Diagnosed Source File To Library



ENTER NUMBER OF OPTION REQUIRED -->
```

Screen 2

```
                    COBOL PROCEDURE                    OPTIONAL-(O)

                  Compiles a COBOL Program.

Name Of Source Program To Be Compiled .....
Name Of Source Input Library ..... $LIBRARY
Name Of Object Output Library ..... $LIBRARY
Name Of Blocks For Compiler Work Files (1-9999) ..... 24
Maximum Number of Requesting Terminals (0-255) ..... 0
Never Ending Program (YES/NO) ..... NO
Name Of File To Receive Merged Source And Diagnostics .....      (O)
Place On Input Job Queue (YES/NO) ..... NO
```

Screen 3

```
                    COBOLG PROCEDURE                   OPTIONAL-(O)

                  Executes a COBOL Program.

Name Of Program To Be Executed .....
Name Of Procedure Containing User OCL .....      (O)
Name Of Library Containing Program To Be Executed .....      (O)
Place Job On The Input Job Queue (YES/NO) ..... NO
```

Screen 4

```
                    COBOLCG PROCEDURE                  OPTIONAL-(O)

                Compiles And Executes a COBOL Program.

Name Of Source Program To Be Compiled .....
Input Library For Source Member ..... $LIBRARY
Output Library For Load Member ..... $LIBRARY
Number Of Blocks For Work Files (1-9999) ..... 24
Maximum Number Of Requesting Terminals (0-255) ..... 0
Never Ending Program (YES/NO) ..... NO
Name Of Procedure Containing OCL .....      (O)
Name Of File To Receive Merged Source And Diagnostics .....      (O)
Name Of Library Containing Program To Be Executed .....      (O)
Place On Input Job Queue (YES/NO) ..... NO
```

Screen 5

```
                    COBOL SEU PROCEDURE

                Creates or Updates a COBOL Module.

Name Of Module To Create Or Update .....
Module Type (S for Source, P for Procedure) ..... S
Format Member Name ..... $SEQXTRA
Name Of Library Containing Module ..... $LIBRARY
```

Screen 6

```
                    COBMOVE PROCEDURE

                Moves a Diagnosed Source File To a Library.

Label For The Diagnosed Source File .....
Output Library Name ..... $LIBRARY
To Replace An Existing Member, Enter REPLACE .....      (O)
To Remove Source File, Enter DELETE .....      (O)
```

Figure 9-2. COBOLP Command Statement Screen Prompts

PROCESS STATEMENT

The PROCESS statement allows you to specify compile-time options unique to COBOL. The PROCESS statement must be placed before the first source statement in the COBOL program immediately preceding the IDENTIFICATION DIVISION header.

The format of the PROCESS statement is as follows:

```
PROCESS option-1 [option-2] . . . [option-n]
  [COPY statement.]
```

The following rules apply to the PROCESS statement:

1. The word PROCESS and all options must appear within positions 8 through 72. Position 7 must be left blank. The remaining positions can be used as in COBOL source statements, positions 1 through 6 for sequence numbers, positions 73 through 80 for identification purposes.

2. Options must be separated by one or more blanks and/or commas.

3. Options may appear in any order. If conflicting options are specified, for example LINK and NOLINK, the last option encountered takes precedence.

4. The PROCESS statement begins with the word PROCESS. Options may appear on more than one line; however, only the first line may contain the word PROCESS.

The following list identifies and describes PROCESS statement options. The underscore indicates the default for each option.

SOURCE NOSOURCE	SOURCE causes the compiler to print the COBOL source statements. NOSOURCE suppresses the SOURCE option.
MAP NOMAP	MAP causes the compiler to print a Data Division map. No specification or NOMAP suppresses the MAP option.
LIST NOLIST	LIST causes the compiler to print a Procedure Division map. It also directs the linkage editor to print a link-edit map if the LINK option is specified. No specification or NOLIST suppresses the LIST option.
FLAGE FLAGW	FLAGW indicates that all diagnostics are to be listed (severity levels W, C, and E). FLAGE indicates that only those diagnostics with a severity level of E are to be listed.
QUOTE APOST	QUOTE indicates to the compiler that the double quotation mark (") should be accepted as the character to delineate literals. APOST indicates that the apostrophe (') should be accepted. This option also specifies the character to be generated for the figurative constant QUOTE(S).
LIBRARY(libname)	LIBRARY controls the processing of COBOL COPY statements by specifying the location of the source library. It is used only if IN/OF libname is not specified on the COPY statement. libname specifies the System/34 library that contains the member to be copied. If the LIBRARY option is not specified, the system library is assumed.
CMPAT(COMP) NOCMPAT	CMPAT(COMP) specifies that COMPUTATIONAL(COMP) data items are to be encoded in the binary data format used on System/370. No specification or NOCMPAT suppresses the CMPAT option. **CAUTION** This option is to be used only when compatibility is essential. Less efficient code is generated when the CMPAT option is specified.

LVL(A/B/C/D)
NOLVL

Specifies what level of FIPS flagging is to be used. If flagging is specified, source clauses and statements that do not conform to the specified level of FIPS are identified.

All FIPS flagging is done according to the 1975 FIPS standard.

The corresponding FIPS levels flagged are:
A = Low
B = Low-Intermediate
C = High-Intermediate
D = Full FIPS COBOL flagging

OBJECT
(R/P [,LIB (libname)])
NOOBJECT

OBJECT specifies that the nonexecutable object module created by the compiler is to be saved in a library after compilation is completed.

R (replace) or P (prompt) specifies the disposition of the module. If this disposition is omitted, a replace disposition is assumed.

If disposition is P and the module exists in the library, the system displays a message and the operator decides if the duplicate member is to be replaced.

LIB(libname) specifies the name of the library where the object module is to be placed. If this is omitted, the module is placed in the library specified by the OUTLIB parameter on the COMPILE OCL statement. If OUTLIB is omitted, the module is placed in the system library. No specification or NOOBJECT suppresses the OBJECT option.

LINK
(R/P [,LIB (libname)])
NOLINK

LINK specifies that link-editing is to be performed and that the executable object module is to be saved in a library after compilation is completed.

R (replace) or P (prompt) specifies the disposition of the module. If this disposition is omitted, a replace disposition is assumed.

If disposition is P and the module exists in the library, the system displays a message and the operator decides if the duplicate member is to be replaced.

LIB(libname) specifies the name of the library where the load module is to be placed. If this entry is omitted, the module is placed in the library specified by the OUTLIB parameter on the COMPILE OCL statement, the COBOL command statement, or the COBOLCG command statement. If OUTLIB is omitted, the module is placed in the system library.

NOLINK specifies that no executable load module is to be created. If neither LINK nor NOLINK is specified, link-editing is performed, and the executable module is placed in the library specified by the OUTLIB parameter on the COMPILE OCL statement, the COBOL command statement, or the COBOLCG command statement. If the OUTLIB parameter is omitted, the module is placed in the system library.

LET
NOLET

LET suppresses the halt if error message CBL 1019 is issued (C or E level diagnostics detected). No specification or NOLET suppresses the LET option.

SEQ NOSEQ	SEQ indicates that the compiler is to check the sequence of the source module statements. If the statements are not in sequence, a message is printed.
SYNTAX NOSYNTAX	SYNTAX causes the compiler to perform only syntax checking with absolute suppression of object code generation, and to produce appropriate error messages. NOSYNTAX causes normal compilation, with both syntax checking and object code generation.
XREF NOXREF	Indicates whether or not a sorted cross-reference listing is produced. If XREF is specified, a listing is produced with data-names and procedure-names appearing in two parts in sorted order.
SUBLIB(libname)	libname is the library from which user-supplied subroutines are obtained by the overlay linkage editor.
NOHALT HALT	Specifies whether compiler errors should cause a halt or a message. For these errors, a halt will be issued unless NOHALT is specified. For messages CBL–1000 to CBL–1099, NOHALT suppresses the halt and output to the display station (that is, output is to the printer only). In addition, the compiler assumes a 0 response and continues compilation if it encounters errors CBL–1021 or CBL–1022.
LIB NOLIB	Indicates whether or not COPY processing will be needed for this compilation. If no COPY statements appear in the source to be compiled, the specification of NOLIB saves compilation time.
GRAPHIC NOGRAPH	Indicates whether or not ideographic literals are present in the COBOL program. For COBOL to process ideographic literals, you must have the ideographic version of the SSP. For more information on ideographic support see Chapter 10, *Ideographic Support*.
SORT(PRINT / NOPRINT)	Indicates to the System/34 sort utility whether or not to print its output when the COBOL SORT statement is executed.

Using COPY Within the PROCESS Statement

The COBOL COPY statement can be used within the PROCESS statement to retrieve compiler options previously stored in a source library and include them in the PROCESS statement. If the LIBRARY option was specified, the requested user library is searched for the source member. If the LIBRARY option was not given, the system library is searched. COPY can be used, to include options that override those specified as default by the compiler. Any PROCESS statement options may be retrieved with the COPY statement.

Note: If you are copying PROCESS statement options into your program, the REPLACING option of the COPY statement is not operational until the PROCESS statement is finished.

Compiler options may both precede and follow the COPY statement within the PROCESS statement. The last encountered occurrence of an option overrides all preceding occurrences of that option.

The following example shows the use of the COPY statement within the PROCESS statement. The COPY statement must be followed by a period. Notice also that in this example, NOMAP and NOOBJECT override the corresponding options in the library member:

```
000001  PROCESS QUOTE                                  MYPROG
000002  COPY DEFLTS.                                   MYPROG
        MAP, SOURCE, OBJECT (R,LIB(MYLIB)), LIST       DEFLTS
000004  NOMAP, NOOBJECT FLAGW                          MYPROG
000010  IDENTIFICATION DIVISION.                       MYPROG
```

THE USER LIBRARY

The user library is an area on disk for storing procedures and source statements. Procedures are groups of OCL statements used to load programs. Source statements are sets of data, such as COBOL source programs.

Storing Procedures and Source Statements

To be retrieved from the user library, procedures and source statements must first be placed in the library. Statements may be stored in a library by the library maintenance program.

The copy function of the library maintenance program has the ability to add or replace a source or procedure member in the library. This function is particularly useful because precoded OCL procedures and sets of COBOL source statements can be stored and made available for future retrieval.

A complete description of the copy function and the other functions of library maintenance (allocate, delete, and rename) can be found in the *System Support Reference Manual.* Also see the *Source Entry Utility Reference Manual* for information on entering source statements.

Retrieving COBOL Source Statements

The System/34 COBOL compiler can copy a source module from a user library into a COBOL program being compiled. A COBOL source program can be composed of a combination of uniquely coded source statements, as well as source statements incorporated into a program at the time of compilation from the user library. The presence of the COBOL COPY statement indicates that source statements are to be included in the source program.

The library member from which text is copied is determined as follows:

1. The library specified on the COPY statement takes precedence.

2. The library specified by the LIBRARY option of the PROCESS statement takes next higher precedence.

3. The system library (#LIBRARY) is the default if neither of the above is specified.

Retrieving an Entire COBOL Source Program

The System/34 COBOL compiler can compile an entire source program directly from the user library instead of from the system input device. The presence of the OCL COMPILE statement in the job stream indicates that the library member identified by the SOURCE parameter is to be compiled.

All COBOL statements (including the COPY statement) are allowed in the source library member. The programmer is responsible for placing the source program in the library. If special compiler options are required, a PROCESS statement should precede the source program in the library.

LINK-EDITING

Link-editing is the process of transforming nonexecutable object modules (compilation output) into executable load modules. That is, link-editing prepares the output of compilation for execution.

When the program to be executed consists only of the output of a single compilation, the programmer can use the LINK option of the PROCESS statement to request link-editing at compile time, either explicitly or by default. The LINK option indicates that link-editing is to be performed using the nonexecutable object modules created by the compiler in the compilation just completed. Output from the link-edit function is placed in the object library. The programmer must specify the necessary LINK option parameter to obtain output on the library of his choice. The LINK option is discussed in detail under *PROCESS Statement* earlier in this chapter.

When the program to be executed consists of nonexecutable object modules created as a result of several separate compilations, these nonexecutable object modules must be transformed into executable load modules by the overlay linkage editor.

The overlay linkage editor is discussed in detail under *Link Editing with Overlay* later in this chapter.

EXECUTION

The load module execution executes a COBOL program that has been compiled and merged with other object modules into a load module.

Load module input consists of either COBOL-supplied procedures or OCL statements defining the step, and any program data to be processed by the load module. Output consists of program output and execution messages.

Execution of a module is requested by the presence of an OCL LOAD statement followed by an OCL RUN statement in the job stream. The location is determined by the parameter specified in the LINK option. If neither LINK nor NOLINK is specified on the PROCESS statement, link-editing is the default. The name of the object program must also be specified on the OCL LOAD statement. This name is the first six characters of the name specified in the PROGRAM-ID paragraph of the source program.

In the following example of a procedure, assume that the name of the procedure is PROC1. The procedure-name is the name that identifies the procedure in the library.

```
// LOAD ENDMON
// FILE NAME-DALTOT,UNIT-F1,RECORDS-1500,
//   RETAIN-P
// FILE NAME-ACCTOT,LABEL-TOTAL,UNIT-F1,
//   DATE-1/04/79
// SWITCH XXX01XX0
// RUN
```

To use this procedure unchanged in the job stream requires the following statement:

```
PROC1
```

The most common OCL statements used for object module execution are listed and discussed in Figure 9-3. A complete description of OCL statements can be found in the *System Support Reference Manual*.

Statement	How Used
// DATE	Supplies the system with the program *date*. The ACCEPT statement with the FROM DATE option can be used to move the date to a user-specified location where it is accessible to the programmer.
// LOAD	Identifies the load module (object program) to be executed.
// FILE	Supplies the system with information about disk files. A FILE statement must be supplied for each disk file created or processed by the COBOL program during the execution job step. (For an SD file, a FILE statement is optional, except when input procedures are used. When input procedures are used, the SD file must have a FILE statement with RETAIN-S.) The NAME parameter specifies to the system the file-name used in the ASSIGN clause in the program. It must be placed on the first line of the FILE statement. The LABEL parameter specifies the name by which the file is identified on disk. If the LABEL parameter is missing, the name in the NAME parameter is substituted.
// SWITCH	Sets any or all of the eight external indicators. These switches are known to the COBOL programmer as the function-names UPSI-0 through UPSI-7. Condition-names may be assigned to the ON or OFF status of these function-names in the SPECIAL-NAMES paragraph of the Environment Division. These condition-names can then be tested in the Procedure Division.
// RUN	Indicates the end of the OCL statements for a program. After the system reads a RUN statement, it loads and executes the program specified on the OCL LOAD statement.
/*	Indicates the end of any input data being read from the SYSIN (system input) device.

Figure 9-3. Statements Used for Object Program Execution

PROGRAM LINKAGE

Whenever a program calls another program, linkage must be established between the two. The calling program must state the name of the called program and must specify any data to be passed. The called program accepts the data and must establish linkage for the return of control to the calling program.

This section describes the accepted linkage conventions for calling and called programs written in COBOL, and the operation of the overlay linkage editor with COBOL programs. It also describes the overlay structure that enables different called programs to occupy the same area of main storage at different times.

Calling and Called Programs

A program that passes control to another program is a *calling program.* A program that receives control from a calling program is a *called program.* Such programs must be compiled separately, and the resulting nonexecutable object modules must then be link-edited by the overlay linkage editor, so as to produce a single executable load module.

A called program can also be a calling program; that is, a called program can, in turn, call another program. In Figure 9-4 for example, program A calls program B; program B calls program C. Therefore:

1. A is considered a calling program by B.
2. B is considered a called program by A.
3. B is considered a calling program by C.
4. C is considered a called program by B.
5. B is both a calling and a called program.

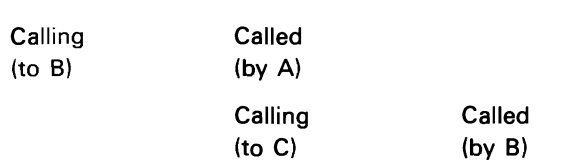


Figure 9-4. Calling and Called Programs

In System/34 COBOL, a called program can call any other program except one that has directly or indirectly already called it. In Figure 9-4, for example, program A can call program B or program C, and program B can call program C. Program C, however, cannot call program B (because B called it directly); neither can it call program A (because A called it indirectly via B). If program C called program B, program B would be unable to return to program A, and a loop might result.

Called programs have the option of terminating the entire program or of returning to their respective calling programs. The STOP RUN statement is used to terminate execution; the EXIT PROGRAM statement is used to return to the calling program. In Figure 9-4, program A is not a called program; it must issue a STOP RUN statement at its completion. An EXIT PROGRAM statement in program A would be bypassed at execution time and have no effect on program flow. The use of the STOP RUN and EXIT PROGRAM statements is shown under *Example of Calling and Called Programs with the USING Option* later in this chapter.

In System/34 COBOL, called programs always appear in their last used state. It is the COBOL programmer's responsibility to reinitialize work areas and altered GO TO statements if they are needed in an initial state for proper reexecution. Conversely, the COBOL programmer can change data areas or OPEN files in such programs with the assurance that they will remain that way until the program is again called.

COBOL CALL Statement Linkage

A calling COBOL program must contain a CALL statement at the point where another program is to be called:

```
CALL 'program-name' [USING identifier-1
  [identifier-2] ...].
```

The CALL statement states the name of the called program and must specify any data items (identifiers) that are passed to the called program.

A called COBOL program can contain a USING option in its Procedure Division header:

```
PROCEDURE DIVISION [USING data-name-1
  [date-name-2] ...].
```

The USING option must contain a data-name to correspond to each identifier in the CALL statement of the calling program.

- The number of identifiers must be the same as the number of data-names; if the numbers are unequal, results are unpredictable.
- The identifiers and data-names must be in the proper order; this order is positional–not by name. That is, the first identifier must correspond to the first data-name, the second to the second, and so on.
- Corresponding identifiers and data-names must be in the same data-format. One way to do this is to place all data descriptions that will be used as identifiers or data-names in the source library and include them in both calling and called programs with the COBOL COPY statement.

A called COBOL program must also contain an EXIT PROGRAM statement if control is to be returned to the calling program. This statement returns control to the calling program at the point immediately following the CALL statement.

For a complete description of the CALL statement and the USING option, see *CALL Statement* in Chapter 6.

Identifier References in a Called Program

When a COBOL CALL statement is executed, the calling program constructs a list of the addresses of the identifiers specified in the CALL statement. When the called program is entered, it saves this list. All subsequent references in the called program to Linkage Section items are made based on the addresses contained in this list. Such references are made directly to the actual data area in the calling program. The contents of an identifier can be changed before returning from a called to a calling program.

For example, if the programmer executes a MOVE operation from one Linkage Section item to another, data is moved from one location to the other in the calling program.

No space is set aside for data items that appear in the Linkage Section; these items are only used to describe the format of the CALL identifiers to the COBOL compiler.

Calling and Called Programs with the USING Option

Figure 9-5 shows two sets of source statements: one for the calling program, CALLPG, and another for the called program, SUBPRG.

Processing begins in CALLPG, which is the calling program. When the statement

```
CALL 'SUBPRG' USING RECORD-1
```

is executed, control is transferred to the first statement of the Procedure Division in SUBPRG, which is the called program. In the calling program, the identifier of the USING option is RECORD-1.

When CALLPG tranfers control to SUBPRG, the values within RECORD-1 are made available to SUBPRG. Within SUBPRG, however, they are referred to as PAYREC. The PICTURE clauses for the subfields of PAYREC (described in Linkage Section of SUBPRG) are the same as those for RECORD-1.

When processing within SUBPRG reaches the EXIT PROGRAM statement, control is returned to CALLPG at the statement immediately following the original CALL statement. Processing then continues in CALLPG until the STOP RUN statement is reached. The calling program is then terminated.

In any given execution of these two programs, if the values within RECORD-1 are changed between the time of the first CALL and any reissuing of the CALL, the values passed at the time of the second CALL will be the changed, not the original, values. If the programmer wishes to use the original values, then he must ensure that they have been saved.

Calling Program	Called Program
IDENTIFICATION DIVISION. PROGRAM-ID. CALLPG. • • • DATA DIVISION. • • • WORKING-STORAGE SECTION. 01 RECORD-1. 03 SALARY PICTURE S9(5)V99. 03 RATE PICTURE S9V99. 03 HOURS PICTURE S99V9. • • • PROCEDURE DIVISION. • • • CALL 'SUBPRG' USING RECORD-1. • • • STOP RUN.	IDENTIFICATION DIVISION. PROGRAM-ID. SUBPRG. • • • DATA DIVISION. • • • LINKAGE SECTION. 01 PAYREC. 02 PAY PICTURE S9(5)V99. 02 HOURLY-RATE PICTURE S9V99. 02 HOURS PICTURE S99V9. • • • PROCEDURE DIVISION USING PAYREC. • • • EXIT PROGRAM.

Figure 9-5. Example of Source Statements for Calling and Called Programs

Link-Editing of Calling and Called Programs

Because calling and called programs are compiled separately, they result in nonexecutable object modules that must be converted to a single executable load module by link editing.

Link-editing is the process of preparing the output of language translators for execution. The program used for this process is the overlay linkage editor. The overlay linkage editor performs the following functions:

- Combines separately produced object modules
- Resolves symbolic cross-references among object modules
- Replaces, deletes, and adds control sections
- Produces a load module that is ready to be fetched into main storage

The overlay linkage editor can operate with or without an overlay structure.

External-Names and References

When providing linkage for COBOL programs, the overlay linkage editor resolves external-names and external references and combines calling and called programs.

An external-name is a name that can be referenced by another program that has been separately compiled and assembled. For each COBOL program, an external-name is created from the program-name. An external reference is a reference that one program makes to an external-name in another program. In COBOL, an external reference is created for each CALL statement (from the program-name field). External-names and references are listed in an overlay map produced by the overlay linkage editor, which can be used by the programmer to check the calling paths if a link-edited program fails to operate properly.

Link-Editing Without Overlay

This section discusses the operation of the overlay linkage editor when no overlay structure is specified. For example, assume that a COBOL main program, CBMAIN, at one or more points in its logic, executes CALL statements to COBOL programs SUBPRA, SUBPRB, and SUBPRC. Also assume that the module sizes for main program, subprograms, and required object-time subroutines are:

Program	Module Size (in bytes)
CBMAIN	5000
SUBPRA	1000
SUBPRB	1500
SUBPRC	2500
Subroutines	2000

No overlay structure need be specified at link-edit time if 12000 bytes (the sum of the module sizes) or more are available to execute the program. Usually, COBOL programs run faster when no overlay structure is specified. In general, programmers should not use the overlay structure except when the sum of the module sizes exceeds the size of available storage. (Sometimes, however, overlay structures result in a much faster object program. These cases are discussed under *Link-Editing with Overlay* later in this chapter.)

Figure 9-6 is an example of the OCL and COBOL statements needed to compile and link-edit the previously mentioned calling and called programs without specifying an overlay structure. The example shows the compilation of CBMAIN, SUBPRA, and SUBPRB, yielding nonexecutable object modules that are cataloged permanently in the library, MYLIB. It is further assumed that the nonexecutable object module for SUBPRC is already in MYLIB. The PROCESS statement LINK option causes the link-editing of these modules into an executable load module, cataloged permanently under the name CBMAIN in the system library. Only the critical portions of each source program are illustrated. Finally, the steps required for execution of the object program are shown.

```
COBSYSIN
        PROCESS OBJECT(P,LIB(MYLIB)),NOLINK
        IDENTIFICATION DIVISION.
        PROGRAM-ID. SUBPRA.
        .
        .
        DATA DIVISION.
        .
        .
        WORKING-STORAGE SECTION.
        01 SUB-ARGUMENT PIC XX.
        .
        .
        PROCEDURE DIVISION.
        .
             CALL 'SUBPRB' USING SUB-ARGUMENT
             EXIT PROGRAM. (or STOP RUN.)
/*
COBSYSIN
        PROCESS OBJECT(P,LIB(MYLIB)), NOLINK
        IDENTIFICATION DIVISION.
        PROGRAM-ID. SUBPRB.
        .
        .
        DATA DIVISION.
        .
        .
        LINKAGE SECTION.
        77 MY-ARGUMENT PIC XX.
        .
        .
        PROCEDURE DIVISION USING MY-ARGUMENT
        .
             EXIT PROGRAM. (or STOP RUN.)
/*
COBSYSIN
     PROCESS OBJECT(P,LIB(MYLIB)),LINK(P),
     SUBLIB(MYLIB)
     IDENTIFICATION DIVISION.
     PROGRAM-ID. CBMAIN.
     ENVIRONMENT DIVISION
     .
     .
     OBJECT-COMPUTER. IBM-S34
          MEMORY SIZE 12000 CHARACTERS.
     .
     .
     DATA DIVISION.
     .
     .
     WORKING-STORAGE SECTION.
     77 MAIN-ARGUMENT PIC XX.
     .
     .
     PROCEDURE DIVISION.
          CALL 'SUBPRC'.
          CALL 'SUBPRA'.
          STOP RUN. (EXIT PROGRAM statements
            have no effect in main programs)
/*
```

Figure 9-6 (Part 1 of 2). Example of OCL and COBOL Statements for Link-Editing Without Overlay

```
***** EXECUTE THE OBJECT PROGRAM *****
// LOAD CBMAIN
    [// FILE Statements, if required]
// RUN
    [input data, if required]
/*
```

Figure 9-6 (Part 2 of 2). Example of OCL and COBOL Statement for Link-Editing Without Overlay

Figure 9-7 is an example of the OLINK procedure needed to link-edit the previously mentioned calling and called programs without specifying an overlay structure. The generated OCL would be used if all programs had been previously compiled and placed in the object library as nonexecutable object modules (either by the OBJECT option or by the $MAINT utility program).

```
OLINK CBMAIN,MYLIB,,,,,MYLIB, #COBLIB
```

Figure 9-7. Example of the OLINK Procedure for Link-Editing Without Overlay, when all Modules Are in the Object Library

Link-Editing with Overlay

If a program is too large to be contained in the main storage available at execution time, it can still be executed by means of an *overlay structure.* (The COBOL segmentation feature can also be used. This feature is described in detail under *Segmentation Feature* in Chapter 6.)

An overlay structure permits different called programs to alternately occupy the same storage area. Overlay structures are created by the overlay linkage editor under the direction of the programmer. For further information, refer to the *Overlay Linkage Editor Reference Manual.*

Not all programs can be link edited to form overlay structures. This can be done only if the sum of the sizes of those programs that must be in main storage simultaneously does not exceed the size of main storage available for execution. The location of the COBOL CALL statements (that is, the logic of the programs) determines which programs must be in storage simultaneously.

From the CALL statements, a simple diagram known as a tree can be constructed. All programs that are called from any given calling program must be added to the tree immediately below that calling program. A diagram of the tree structure of the programs in Figure 9-6 is given in Figure 9-8.

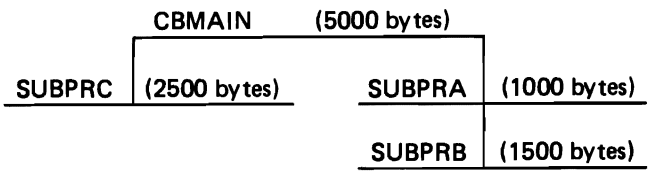


Figure 9-8. Tree Structure Representing Programs Link-Edited with Overlay

COBOL object-time subroutines are not shown on the tree structure in Figure 9-8. They do, however, increase the size of the final overlay program. The assumption will be made that the object-time subroutines occupy 2000 bytes.

Once the tree structure of a given program has been put into the form of a diagram, it is relatively easy to determine whether a program will fit in the main storage available. Simply trace each possible path through the tree, from calling program to called program (downward), and add the sizes of the modules encountered plus the amount required for object-time subroutines. The largest of these sums is the minimum number of bytes that must be available for the overlay program.

For example, the tree structure in Figure 9-8 has two paths, both of which yield a sum of 7500 bytes. Thus, the overlay program requires at least 9500 bytes (2000 bytes of which are for object-time subroutines). The program would then be eligible for overlay in 9500 or more bytes of main storage but not in less than 9500 bytes of main storage.

Making Subprograms Eligible for Overlay: In order to ensure that a subprogram is eligible for overlay, the following guidelines must be followed:

- If the called program contains file description entries (FDs), files should be opened and closed before an EXIT PROGRAM statement is executed. This eliminates the possibility of the calling program issuing a CALL to a subprogram in another overlay and, therefore, destroying an open file.

- If the called program requires information to be saved for the next time it is called, the information must be saved in an area that is safe from overlay. One method is to define an area in the main program that is used exclusively for this purpose. The address of this common area would be passed from calling to called programs (preferably always as the first identifier). The COBOL COPY statement would be used to include its data description in the Linkage Section of all subprograms as well as in the Working-Storage Section of the main program.

Determining the Overlay Stucture: The overlay linkage editor determines the overlay structure of the overlay program, based on the following:

- The tree structure of the program

- The amount of storage that will be available at execution time

- The linkage editor CATEGORY statement(s)

- The linkage editor GROUP statement(s)

The tree structure is determined by the location of the CALL statements and the sizes of the called programs.

The amount of storage available at execution time must be specified by the programmer. It is taken from the CORE parameter of the linkage editor OPTIONS statement (or COBOL MEMORY SIZE clause if the COBOL compiler is invoking the linkage editor). If storage size is omitted, the size of the current region is used.

The linkage editor CATEGORY statements must be provided by the programmer. The programmer must specify a category of 8 through 126 for each subprogram eligible for overlay (categories 1 through 7 are reserved for system use). A low category number increases the chance that the overlay linkage editor will make a module resident if there is enough space to do so. A high category number decreases that chance. Thus, the programmer should assign low numbers to small programs that are called frequently, high numbers to large programs that are called infrequently, and numbers in the middle for programs that are of mixed type.

The linkage editor GROUP statement can optionally be provided by the programmer. Without this statement, the overlay linkage editor constructs an overlay structure based on a generalized algorithm that is designed to provide good performance over a wide range of programs. The programmer may, however, be able to improve on this algorithm by instructing the linkage editor, with GROUP statements, as to which modules it should group together. A GROUP statement should be used whenever two or more called programs are frequently used together but do not call each other.

The OPTIONS, CATEGORY, and GROUP statements are described in detail in the *Overlay Linkage Editor Reference Manual*.

Statements for Accomplishing Overlay: Assume that the jobs in Figure 9-6 have been run and the programmer needs to reduce the size of the executable load module to allow it to execute in 10 K bytes of main storage (1 K = 1024 bytes). Further assume that SUBPRA, SUBPRB, and SUBPRC are all eligible for overlay. The linkage editor statements illustrated in Figure 9-9 will construct an overlay program that executes in 10 K bytes. The categories specified in the illustration have no relevance because there is only one overlay structure of the given program that will fit in 10 K bytes of main storage.

```
// LOAD #OLINK
// FILE NAME-$SOURCE,...   (same as in
// FILE NAME-$WORK,...     COBOL procedure)
// RUN
// PHASE NAME-CBMAIN
// OPTIONS STORE-10 K
// MODULE NAME-CBMAIN
// CATEGORY NAME-SUBPRA,VALUE-20
// CATEGORY NAME-SUBPRB,VALUE-20
// CATEGORY NAME-SUBPRC,VALUE-40
// END
```

Figure 9-9. OCL for Accomplishing Overlay

When Overlays Are Beneficial: Under certain circumstances, overlay programs (as well as COBOL programs that use the segmentation feature) can be faster than nonoverlay programs. Consider the case of a program designed to read a large disk file, recording running statistics, then printing this information in the form of a graphic chart. Assume that the graphic output is complex enough to require a large amount of code. In this situation, an overlay structure should be created; one program to do the printing, a second program to read disk and calculate. This structure allows larger buffer size for the disk access portion, thus potentially increasing execution speed.

Interfacing with the Overlay Linkage Editor

There are two possible ways of interfacing with the overlay linkage editor:

1. The System/34 COBOL compiler can interface with the overlay linkage editor for the programmer.

2. The programmer can interface directly with the overlay linkage editor by using a stand-alone link or a separate link step.

The overlay linkage editor uses the current region size or the value of the MEMORY SIZE clause when the programmer interfaces through the System/34 COBOL compiler. However, the programmer can also specify a larger or smaller region size if he performs a separate link step as mentioned previously. For additional information, see the *Overlay Linkage Editor Reference Manual.*

LINKAGE BETWEEN MODULES PRODUCED BY SYSTEM/34 LANGUAGE TRANSLATORS

This section describes standard linkage conventions for use between modules produced by the System/34 language translators: COBOL, FORTRAN IV, and Assembler. Programmers using standard linkage conventions are able to code routines in the language most appropriate to the function being performed. Figure 9-10 illustrates the standard linkage convention described on the following pages.

```
*     ASSEMBLER MODULE (MODA) CALLS COBOL MODULE (MODB)
*
          EXTRN MODB
MODA      START X'0000'
*
*
*
*
          B     MODB              CALL COBOL MODULE MODB
          DC    AL2(PLIST)        PARAMETER LIST
*                                 CONTROL RETURNS HERE AFTER MODB EXECUTION
*
*
*
*
*
*
*                                 PARAMETER LIST
PLIST     EQU   *
          DC    AL2(SAVA)         ADDRESS OF SAVE AREA
          DC    AL2(PARM1)        ADDRESS OF FIRST PARAMETER.
          DC    AL2(PARM2)        ADDRESS OF SECOND PARAMETER
*
*
*
          DC    XL2'FFFF'         END OF PARAMETER LIST INDICATOR
*
*                                 PARAMETERS
PARM1      EQU  *
          DC    CL5'FIRST'
PARM2     EQU   *
          DC    CL6'SECOND'
*
*
*
*                                 SAVE AREA
SAVA      DC    XL1'B0'           INDICATOR BYTE - CALLING PROGRAM IS ASSEMBLER
          DC    CL6'MODA'         CALLING PROGRAM'S NAME
          END   MODA
```

```
*         SAMPLE SYSTEM/34 LINKAGE
*
*         COBOL MODULE (MODC) CALLS ASSEMBLER MODULE (MODD)
*
XR1       EQU   1                                   EQUATE REGISTER VALUES
XR2       EQU   2
ARR       EQU   8
IAR       EQU   16
*
          ENTRY MODD
MODD      START X'0000'
          ST    SAVAR1,XR1                          SAVE INDEX REGISTER ONE VALUE
          LA    SAVA,XR1                            POINT XR1 TO MODD'S SAVE AREA
          USING SAVA,XR1                            ESTABLISH XR1 AS BASE REGISTER
          ST    SAVAR2(,XR1),XR2                    SAVE INDEX REGISTER TWO VALUE
          ST    SAVART(,XR1),ARR                    SAVE ARR VALUE
          L     SAVART(,XR1),XR2                    POINT XR2 TO ARR VALUE
          L     1(,XR2),XR2                         POINT TO SECOND BYTE OF ADDRESS
          ALC   SAVART(,XR1),TWO(,XR1)              JUMP ARR VALUE PAST PARAMETER
          L     3(,XR2),XR1                         GET ADDR OF PARAM 1 INTO XR1
          L     5(,XR2),XR2                         GET ADDR OF PARAM 2 INTO XR2
*
*         BODY  OF  ROUTINE. . .
*         RETURN TO CALLING PROGRAM
*
          L     SAVAR2(,XR1),XR2                    RESTORE XR2 VALUE
          L     SAVAR1(,XR1),XR1                    RESTORE XR1 VALUE
          L     SAVART,IAR                          BRANCH TO NEXT SEQ. INSTRUCTION
*         SAVE AREA
SAVA      DC    XL1'30'
          DC    CL6'MODD'
SAVAR1    DC    XL2'00'
SAVAR2    DC    XL2'00'
SAVART    DC    AL2(00)
*         WORK VALUE
TWO       DC    IL2'2'
          END   MODD
```

Figure 9-10. Standard Linkage

Standard Linkage

Standard Linkage is accomplished as follows:

1. Each module must have a save area defined as follows:

   For a subprogram:

Byte 0	Bit 0	0	Not a main program
	Bits 1-3	000	FORTRAN IV
		001	COBOL
		011	Assembler
	Bits 4-7	0000	Reserved
Bytes 1-6			EBCDIC name, left-adjusted
Bytes 7-8			Value of index register 1 (XR1) at entry
Bytes 9-10			Value of index register 2 (XR2) at entry
Bytes 11-12			Return point in calling program

   For a main program:

Byte 0	Bit 0	1	Main program
	Bits 1-3	000	FORTRAN IV
		001	COBOL
		011	Assembler
	Bits 4-7	0000	Reserved
Bytes 1-6			EBCDIC name, left-adjusted

   Note: Main program refers to the program with the highest level of control.

2. Each module that calls another module must have one or more parameter lists defined as follows:

Bytes 0-1	Address of save area in this program
Bytes 2-3	Address of first parameter
Bytes (2n)-(2n+1)	Address of nth parameter
Bytes (2n+2)	XL2'FFFF' to indicate end of parameter list

   Notes:
   1. The first 2 bytes, as well as the end-of-parameter-list indicator (XL2'FFFF') must be present in all parameter lists. If no parameters are to be passed, the parameter list is only 4 bytes long. In this case, bytes 3 and 4 are hex FFFF.
   2. Addresses in parameter lists refer to the first byte (byte with the lowest address) of the item.

3. When control reaches a program entry point the address recall register (ARR) must point to a 2-byte field containing the first byte of the parameter list.

   The assembler language code to call a COBOL subprogram would normally be as follows:

```
        EXTRN   SUBR
        B       SUBR
        DC      AL2(PARAMS)
RETNPT  EQU     *
```

   Note: The pointer to the parameter list points to the left byte of the save area address.

4. Normal return is accomplished by placing in the hardware instruction address register (IAR) a value that is two larger than the contents of the ARR when the program was entered.

5. Index registers 1 and 2 (XR1 and XR2) must be saved upon entry in the called program's save area, and restored at exit.

6. The address recall register need not be restored, but the return address must be determined and placed in the called program's save area.

PROGRAM CHECKOUT

COBOL degugging is best accomplished as follows:

- Resolve all compiler diagnostics by eliminating the cause of all conditional (C) or error (E) level messages, and by ensuring that all warning (W) level messages are consistent with the desired result (refer to Appendix A, *Compiler Messages*.

- Resolve all linkage editor diagnostics by correcting the option or external reference, as discussed earlier in this chapter.

- Execute the program.

- Resolve all system and/or COBOL execution-time messages or abonormal terminations. (For debugging, see *Debugging Lines, TRACE Statement, EXHIBIT Statement*, and the discussions of loops, processor checks, and dumps in this chapter. For formats of the TRACE and EXHIBIT statements, see Chapter 6.)

  Note: The use of the TRACE and EXHIBIT statements will greatly increase the amount of time required to execute the program.

- Compare output with expected results. Use EXHIBIT statements to resolve revealed errors.

- Remove all debugging statements from the debugging program by recompiling after removing the WITH DEBUGGING MODE clause from the program.

Debugging Language

The COBOL debugging language is designed to aid the COBOL programmer in producing an error-free program in the shortest possible time. The sections that follow discuss the use of the various types of debugging language and other methods of program checkout.

Debugging Lines

The user can include debugging lines (any COBOL statment with a D in column 7) in his program to assist in location logic errors. By including the WITH DEBUGGING MODE source clause (essentially a compile-time switch), the statements are made part of the object code and will be executed in line with the rest of the program. Removal of the WITH DEBUGGING MODE clause causes debugging lines to be treated as comments only; they will not be executed. A program containing debugging lines must be syntactically correct in both these modes. (The execution-time option DEBUG/NODEBUG has no control over debugging lines; it only affects USE FOR DEBUGGING declaratives–as explained in the following paragraphs.)

Declarative Procedures—USE FOR DEBUGGING

The USE FOR DEBUGGING feature provides the user with the ability to create his own procedures to examine the internal status of his program during its execution. The USE FOR DEBUGGING statement identifies which program elements it wishes to monitor. COBOL then gives the associated procedure control when these elements are referenced during execution. The procedure also is given access to the DEBUG-ITEM special register, which has been automatically filled with the pertinent current status information.

The USE FOR DEBUGGING procedures can be controlled by two switches: the WITH DEBUGGING MODE source clause for compile-time, and the object-time switch for execution-time. WITH DEBUGGING MODE indicates that the procedures are to be compiled as executable code; if WITH DEBUGGING MODE is omitted, the procedures are treated only as comments.

At execution time, a prompt message CBL-3026 is issued to set the object-time switch. A response of 0 to the object-time switch prompt indicates that the procedures compiled into the code are to be executed; if a response of 1 is specified, the procedures are bypassed.

The general rules for USE FOR DEBUGGING declarative procedures and the DEBUG-ITEM special register can be found in Chapter 6.

Figure 9-11 shows a simple example of how debugging can be used. The program contains a debugging phrase [1] and a simple declarative [2]. By removing the WITH DEBUGGING MODE clause from the CONFIGURATION SECTION and recompiling, the programmer can disable the debugging declaratives, even though the declarative statements are left in the source program.

The output generated by the program is shown in the figure for an execution with and another without the object-time debug switch on. The output from the program run with debugging on contains the count of records in and out [3]. The output from the program run with debugging off does not contain the record counts [4].

Trace and Exhibit

Two additional debugging language statements are TRACE and EXHIBIT. Either of these statements can be used as often as necessary. They can be interspersed throughout the COBOL source program. See Chapter 6 for examples of the use of TRACE and EXHIBIT.

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S .........IDENTFCN SEQ/NO S

     PROCESS QUOTE
  1  IDENTIFICATION DIVISION.                                              00010
  2  PROGRAM-ID.  TESTDB.                                                  00020
  3    AUTHOR. PROGRAMMER NAME.                                            00030
  4    INSTALLATION.  ROCHESTER LABORATORY.                                00040
  5    DATE-WRITTEN.  MARCH 6, 1979.                                       00050
  6    DATE-COMPILED.  04/25/79.                                           00060
  7  ENVIRONMENT DIVISION.                                                 00070
  8  CONFIGURATION SECTION.                                                00080
  9    SOURCE-COMPUTER.  IBM-S34 WITH DEBUGGING MODE. 1                    00090
 10    OBJECT-COMPUTER.  IBM-S34.                                          00100
 11  SPECIAL-NAMES.                                                        00110
 12      REQUESTOR IS OPERATOR.                                            00120
 13  INPUT-OUTPUT SECTION.                                                 00130
 14  FILE-CONTROL.                                                         00140
 15      SELECT FILE-1 ASSIGN TO DISK-SAMPLE.                              00150
 16      SELECT FILE-2 ASSIGN TO DISK-SAMPLE.                              00160
 17  DATA DIVISION.                                                        00170
 18  FILE SECTION.                                                         00180
 19  FD  FILE-1                                                            00190
         LABEL RECORDS ARE STANDARD                                        00200
         BLOCK CONTAINS 5 RECORDS                                          00210
         RECORD CONTAINS 20 CHARACTERS                                     00220
         DATA RECORD IS RECORD-1.                                          00230
 20    01 RECORD-1.                                                        00240
 21      02 FIELD-A PICTURE IS X(20).                                       00250
 22  FD  FILE-2                                                            00260
         LABEL RECORDS ARE STANDARD                                        00270
         BLOCK CONTAINS 5 RECORDS                                          00280
         RECORD CONTAINS 20 CHARACTERS                                     00290
         DATA RECORD IS RECORD-2.                                          00300
 23    01 RECORD-2.                                                        00310
 24      02 FIELD-A PICTURE IS X(20).                                       00320
 25  WORKING-STORAGE SECTION.                                              00330
 26  77   KOUNT PICTURE S99 COMP SYNC.                                     00340
 27  77   NUMBER  PICTURE S99 COMP SYNC.                                   00350
 28  77   REC-IN PICTURE 99999.                                            00360
 29  77   REC-OUT PICTURE 99999.                                           00370
 30    01 FILLER.                                                          00380
 31      02 ALPHABET PICTURE X(25) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".     00390
 32      02 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.             00400
 33      02 DEPENDENTS PIC X(26) VALUE "01234012340123401234012340".       00410
 34      02 DEPEND REDEFINES DEPENDENTS PICTURE X OCCURS 26 TIMES.          00420
 35    01 WORK-RECORD.                                                     00430
 36      02 NAME-FIELD  PICTURE X.                                          00440
 37      02 FILLER PICTURE X VALUE IS SPACE.                                00450
 38      02 RECORD-NO PICTURE 9999.                                         00460
 39      02 FILLER PICTURE X VALUE IS SPACE.                                00470
 40      02 LOCATION PICTURE AAA VALUE IS "NYC".                            00480
 41      02 FILLER PICTURE X VALUE IS SPACE.                                00490
```

Figure 9-11 (Part 1 of 3). Example of USE FOR DEBUGGING.

```
STMD -A...B... C O B O L   S O U R C E   S T A T E M E N T S .........IDENTFCN SEQ/NO S

  42       02 NO-OF-DEPENDENTS PICTURE XX.                                  00500
  43       02 FILLER PICTURE X(7) VALUE IS SPACES.                          00510
  44   PROCEDURE DIVISION.                                                  00520
  45   DECLARATIVES.                                                        00530
  46   DEBUG-SECTION SECTION.2                                              00540
  47       USE FOR DEBUGGING ON ALL PROCEDURES.                             00550
  48       IF DEBUG-NAME = "STEP-3" ADD 1 TO REC-OUT.                       00560
  50       IF DEBUG-NAME = "STEP-6" ADD 1 TO REC-IN.                        00570
  52       IF DEBUG-NAME = "STEP-8" EXHIBIT NAMED REC-IN REC-OUT.           00580
  54   END DECLARATIVES.                                                    00590
  55   BEGIN SECTION.                                                       00600
     *     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED     00610
     *     AND INITIALIZES COUNTERS.                                       00620
  56   STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO KOUNT NOMBER.              00630
     *     NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE    00640
     *     CONTAINED IN THE FILE, WRITES THEM ON DISK, AND DISPLAYS        00650
     *     THEM ON THE CONSOLE.                                            00660
  59   STEP-2. ADD 1 TO KOUNT, ADD 1 TO NOMBER, MOVE ALPHA (KOUNT) TO      00670
           NAME-FIELD.                                                     00680
  63       MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.                        00690
  64       MOVE NOMBER TO RECORD-NO.                                       00700
  65   STEP-3. DISPLAY WORK-RECORD UPON OPERATOR. WRITE RECORD-1 FROM       00710
           WORK-RECORD.                                                    00720
  68   STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.      00730
     *     NOTE THAT THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS         00740
     *     INPUT.                                                          00750
  70   STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.                            00760
     *     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES OUT     00770
     *     EMPLOYEES WITH NO DEPENDENTS.                                   00780
  73   STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.     00790
  76   STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO             00800
  79       NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO              00810
           STEP-6.                                                         00820
  81   STEP-8. CLOSE FILE-2.                                               00830
  83       STOP RUN.                                                       00840

  PROGRAM SIZE = DATA DIVISION + PROCEDURE DIVISION + LITERALS + DTF/BUFFERS

     2844              224               799              351          1470


NO ERRORS DETECTED FOR THIS COMPILATION

  END OF COMPILATION

SYS-3130 I TESTDB MODULE'S MAIN STORAGE SIZE IS

           8027 DECIMAL
SYS-3131 I 0000 IS THE START CONTROL ADDRESS OF THIS MODULE
SYS-3134 I TESTDB MODULE IS CATALOGED AS A LOAD MEMBER
           BRCLIB    IS THE LIBRARY NAME
               35 TOTAL NUMBER OF LIBRARY SECTORS
```

Figure 9-11 (Part 2 of 3). Example of USE FOR DEBUGGING.

```
WORK-RECORD = A 0001 NYC Z
WORK-RECORD = B 0002 NYC 1
WORK-RECORD = C 0003 NYC 2
WORK-RECORD = D 0004 NYC 3
WORK-RECORD = E 0005 NYC 4
WORK-RECORD = F 0006 NYC Z
WORK-RECORD = G 0007 NYC 1
WORK-RECORD = H 0008 NYC 2
WORK-RECORD = I 0009 NYC 3
WORK-RECORD = J 0010 NYC 4
WORK-RECORD = K 0011 NYC Z
WORK-RECORD = L 0012 NYC 1
WORK-RECORD = M 0013 NYC 2
WORK-RECORD = N 0014 NYC 3
WORK-RECORD = O 0015 NYC 4
WORK-RECORD = P 0016 NYC Z
WORK-RECORD = Q 0017 NYC 1
WORK-RECORD = R 0018 NYC 2
WORK-RECORD = S 0019 NYC 3
WORK-RECORD = T 0020 NYC 4
WORK-RECORD = U 0021 NYC Z
WORK-RECORD = V 0022 NYC 1
WORK-RECORD = W 0023 NYC 2
WORK-RECORD = X 0024 NYC 3
WORK-RECORD = Y 0025 NYC 4
WORK-RECORD = Z 0026 NYC Z
REC-IN = 00027
REC-OUT = 00026
```

3

```
WORK-RECORD = A 0001 NYC Z
WORK-RECORD = B 0002 NYC 1
WORK-RECORD = C 0003 NYC 2
WORK-RECORD = D 0004 NYC 3
WORK-RECORD = E 0005 NYC 4
WORK-RECORD = F 0006 NYC Z
WORK-RECORD = G 0007 NYC 1
WORK-RECORD = H 0008 NYC 2
WORK-RECORD = I 0009 NYC 3
WORK-RECORD = J 0010 NYC 4
WORK-RECORD = K 0011 NYC Z
WORK-RECORD = L 0012 NYC 1
WORK-RECORD = M 0013 NYC 2
WORK-RECORD = N 0014 NYC 3
WORK-RECORD = O 0015 NYC 4
WORK-RECORD = P 0016 NYC Z
WORK-RECORD = Q 0017 NYC 1
WORK-RECORD = R 0018 NYC 2
WORK-RECORD = S 0019 NYC 3
WORK-RECORD = T 0020 NYC 4
WORK-RECORD = U 0021 NYC Z
WORK-RECORD = V 0022 NYC 1
WORK-RECORD = W 0023 NYC 2
WORK-RECORD = X 0024 NYC 3
WORK-RECORD = Y 0025 NYC 4
WORK-RECORD = Z 0026 NYC Z
```

4

Figure 9-11 (Part 3 of 3). Example of USE FOR DEBUGGING.

Testing a Program Selectively

A debug packet allows the programmer to select a portion of the program for testing. The packet can include test data and can specify operations the programmer wants performed. When the testing is completed, the packet can be removed. The flow of control can be selectively altered by the inclusion of debug packets, as shown in Figure 9-12.

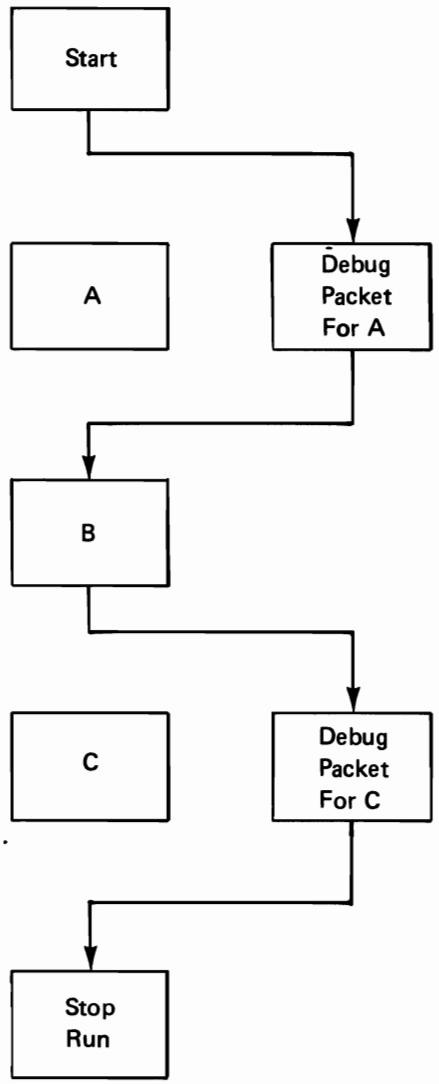


Figure 9-12. Selective Testing of B

In this program, A creates data, B processes it, and C prints it. The debug packet for A simulates test data. It is first in the program to be executed. In the packet, the last statement is GO TO B, which permits A to be bypassed. After B is executed with the test data, control passes to the debug packet for C, which contains a GO TO statement that transfers control to the end of the program, bypassing C.

Testing Changes and Additions to Programs

If a program runs correctly but changes or additions can make it more efficient, a debug packet can be used to test changs without modifying the original source program.

If the changes to be incorporated are in the midddle of a paragraph, the entire paragraph, with the changes included, must be written in the debug packet. The last statement in the packet should be a GO TO statement that transfers control to the next procedure to be executed.

There are usually several ways to perform an operation. Alternative methods can be tested by putting them in debug packets.

The source program library facility can be used for program checkout by placing a source program in a library (see Chapter 6).

PROGRAM LOOPS

When a program repeatedly executes the same series of instructions, and it is apparent that this will continue indefinitely, the program is in a loop. In order to identify loops, it is necessary to make use of information known about the program itself, as follows:

- Time–If the actual run time is substantially exceeding the expected run time, the program may be in a loop.
- I/O actions–If no input/output operations are taking place and I/O is expected to be taking place repeatedly, the program is probably in a loop.

Tracing a Loop in a Program

Frequently, a loop encompasses many instructions in a program. In this case, the programmer must provide COBOL statements to allow the loop to be traced.

The COBOL TRACE statement can be used for this purpose. The simplest method is to execute a READY TRACE statement as one of the first statements in the Procedure Division. From then on, the statement numbers of all paragraphs and sections that are entered will appear on the listing. From this, the programmer can deduce the reason for the loop.

Errors That Can Cause a Loop

The following are examples of COBOL language errors that can cause a loop:

1. A GO TO statement with no procedure-name following it has not been initialized with an ALTER statement. The program will loop at the unaltered GO TO statement. (The IAR will point to the unaltered GO TO statement.) This problem may be avoided by coding:

   ```
   ALT-PARA. GO TO MY-ERROR-ROUTINE.
   ```

   instead of

   ```
   ALT-PARA. GO TO.
   ```

2. A PERFORM with an UNTIL clause is being executed, but the condition specified in the UNTIL clause cannot be met. For example,

   ```
   PERFORM...UNTIL COUNT LESS THAN ZERO
   ```

   where COUNT is an unsigned numeric item.

3. Two PERFORM statements are active at the same time, but they both have the same exit point. For example,

   ```
      GO TO MY-EXECUTION.
   MY-SECTION SECTION.
   MY-FIRST-PARA.
      PERFORM MY-LAST-PARA.
   MY-LAST-PARA.
      DISPLAY 'IN A LOOP'.
   MY-EXECUTION SECTION.
   MY-EXECUTION-PARA.

      PERFORM MY-SECTION.
   ```

   In this example, the execution of PERFORM MY-SECTION establishes return linkage at the end of MY-LAST-PARA, and is then destroyed by the execution of PERFORM MY-LAST-PARA. This results in the reentry of MY-EXECUTION SECTION, causing the loop.

4. A GO TO statement that refers to a previous procedure-name is executed, but no conditional statement exists to prevent the same GO TO statement from being reexecuted. For example,

   ```
   PARA-1.
      MOVE...
      MOVE...
      MOVE...
   PARA-2.
      MOVE...
      GO TO PARA-1.
   ```

A possible variation is that a conditional statements exists, but either the condition cannot be met, or the statement does not branch to (via a GO TO statement) a paragraph outside the range of the loop.

ABNORMAL TERMINATIONS DURING EXECUTION

When the System/34 detects a program condition that prevents continued operaton, an abnormal termination results. Invalid address and invalid operation are the abnormal terminations that are most usually encountered on System/34.

Abnormal terminations are not frequent when compared with the incidence of other problems such as loops and logic errors, and can usually be diagnosed by following certain rules. The following paragraphs describe abnormal terminations.

Abnormal Termination Due to Invalid Address

This type of abnormal termination results when the storage address being referenced is larger than the available region for the user's program. Message SYS-0014 is displayed.

The following usually cause this type of abnormal termination:

- Subscripting or indexing–A field that is being used as a subscript or index contains an invalid value, that is, a value less than 1 or greater than the OCCURS integer for the table. This can cause the calculated address of the table element to be too great, resulting in an invalid address. This problem can be avoided by including the following sentence when referencing tables:

  IF field GREATER THAN occurs-integer OR field LESS THAN 1, PERFORM error-routine GO TO END-OF-JOB.

  Field is the subscript or index being used and occurs-integer is the integer in the OCCURS clause of the table level being referenced.

  Note: If field is defined with USAGE IS INDEX clause, or in the INDEXED BY phrase of an OCCURS clause, it cannot be exhibited or displayed.

- CALL USING...–If the arguments passed in a CALL statement do not match in number, position, and description the arguments in the PROCEDURE DIVISION header, the address calculated for a particular data reference may be too great, resulting in an invalid address. This problem can be avoided only by carefully checking calling and called programs for correspondence of arguments.

Abnormal Termination Due to Invalid Operation

This type of abnormal termination results when the CPU is attempting to execute an instruction but encounters an unrecognizable operation code. Message SYS-0015 is displayed.

Invalid operation can result from the same source errors mentioned for invalid address, since an incorrect address may lead to data overlaying instructions. In addition, the following COBOL error may result in an invalid operation.

In using the segmentation feature, an attempt is made to return from the root to an independent segment that has been overlaid. For example,

```
PROCEDURE DIVISION.
ROOT SECTION 0.
A.    GO TO SEG-50.
B.    PERFORM SEG-60.
C.    STOP RUN.

SEG-50 SECTION 50.
SEG-50-A.
   PERFORM B, GO TO C.
SEG-60 SECTION 60.
SEG-60-A.
   DISPLAY 'SEG-60 ENTERED'.
```

The execution of PERFORM SEG-60 in Paragraph B is invalid, since SEG-50 is in control (via the PERFORM B statement). When Paragraph B attempts to return to SEG-50, it may find it has been overlaid by SEG-60. This will result in either an abnormal termination or an incorrect flow of control.

MAIN STORAGE DUMPS

When a job is abnormally terminated due to a serious error in the program, it may be necessary to examine a dump of main storage to isolate the problem.

Interpreting a Dump

Debugging using dumps is often the fastest method of finding the cause of certain problems, notably loops and abnormal terminations.

The following text describes methods of locating various portions of the COBOL program in a dump. References are made throughout the text to Figures 9-13 through 9-17. The reverse letters in text correspond to reverse letters on the figures that indicate the portion of the listing being discussed.

Note: Many types of bugs are more easily located by using the COBOL debugging language (for example, incorrect results) than examining dumps.

It is recommended that, during the debugging stage, the programmer insert TRACE statements in the program; the presence of any TRACE statement in the program, even if unexecuted, causes COBOL to maintain a statement number list during progrtam execution. This list can be seen in dumps and is a valuable aid in debugging loops and abnormal termination.

Locating the Start of a Program

Addresses in COBOL compiler maps are given relative to the beginning of the program. Thus, to locate data, files, or instructions, it is necessary first to locate the program in which they appear.

The overlay linkage editor provides a map that shows where programs are located in storage. For example, TESTER will be loaded at 0000, as shown by (A) in Figure 9-16.

Determining Paragraphs Executed

The program TESTER purposely contained, at statement 76, a GO TO statement that results in a loop, (B) in Figure 9-13. The range of this loop appears in the dump, because the program contained a TRACE statement (note that RESET TRACE had been executed prior to entering the loop).

The statement number list, which contains statement numbers of the last few paragraphs or sections executed before program termination, contains this information, and can be found as follows:

1. Locate the @CBSTK entry point in the overlay linkage editor map. (It is the @CB510 subroutine.) Note that the address is 111E hex, (C) in Figure 9-16.

2. Refer to location 111E in the dump. Note that the statement number list starts here, and continues for 98 bytes, (D) in Figure 9-17.

   The list is interpreted into readable characters on the right side of the page. It indicates that the paragraph at statement number 76 was being executed and that it was immediately preceded by the one at statement 77 (immediately after the word WERE). Preceding 77 were 76, 77, 76, and so on. Therefore, the loop was occuring within paragraphs 76 and 77, (E) in Figure 9-17.

3. Refer to the COBOL source listing for the source code corresponding to these paragraphs, (B) in Figure 9-13. It is obvious from this that the cause of the loop was the statement

   GO TO STEP-9.

Locating Data

The Data Division map enables the programmer to locate data in the dump. The address in the map must be added to the address in storage at which the phase starts to determine the address of the data area in storage, (F) in Figure 9-14.

For example, the data-name B can be found in the dump as follows:

1. Locate data-name B in the source program. Note that it is in statement 42, (G) in Figure 9-13. (This step is only necessary if you are unsure as to whether another item also has data-name B.)

2. Locate data-name B in the Data Division map, (H) in Figure 9-14. It can be located either by its name (NAME) or by its statement number (STNO). Note that its length (LNTH) is four characters, and that its left-hand address is 0058 hex bytes into the program.

3. Locate the program (PROGRAM-ID. TESTER.) in the overlay linkage editor map. Note that it has been link edited to start at the location 0000 hex, (I) in Figure 9-16. Add the location of B (0058) to the address of TESTER (0000). The data in B may be found starting at address 0058.

4. Locate 0058 in the dump. The field B starts there, and is 4 bytes long. It contains the value 5253571F, (J) in Figure 9-17. For the purposes of demonstration, this is an incorrect result, caused by redefining a field definded as COMP with a field defined as COMP-3.

Locating Files and Buffers

A file may be located in storage in a manner similar to that used to locate data-names.

The FILE-1 file can be found as follows:

1. Locate FILE-1 in the Data Division map, (K) in Figure 9-14. Since it is a file, it has the F flag to make it easy to locate. Note that it starts at 009E hex bytes into the program. (The LNGTH field for files is the logical record length, which is 20; it may be ignored for purposes of this discussion.)

2. Add the start address of TESTER to the displacement of FILE-1. The result is 009E hex, the location of FILE-1 in the dump, (L) in Figure 9-17.

3. Buffers for a file are located after the file (except when a SAME AREA clause is specified for the file). Buffers are allocated in multiples of 256 bytes. Note that buffers can be partially or totally destroyed in the CLOSE process. Such is the case with the FILE-1 buffer, the first portion of which has been overlaid, (M) in Figure 9-17.

Debugging Processor Checks by Dumps

Occasionally, a processor check occurs in program execution. The value of the IAR, ARR, XR1 and XR2 should be recorded. The IAR (instruction address register) or ARR (address recall register) may be pointing to an instruction within the COBOL program. To determine if this is the case, compare their values with the START ADDRESS, until the module name in which the processor check occurred is determined. It might be necessary to refer to the statement number list (see *Determining Paragraphs Executed*).

If it is the COBOL program, subtract its start address from the IAR or ARR value. For example if the IAR value were 0698 then it would be within TESTER, whose start address is 0000. The instruction would be at 0698 into the program.

The statement number can be found by referring to the Procedure Division map. In this case, the statement number is 58. The previous statement should be studied.

Referring to the source listing, statement number 57 can be examined in detail.

Note that the IAR values can be within a statement. In this case, the error occurred specifically in that statement, not in the one preceding.

Note also that the ARR only points to a recently executed instruction. Thus, it should be used only as an aid in determining the cause of a failure.

HINTS FOR PROGRAM CHECKOUT

1. Always specify the following compiler options in the COBOL PROCESS statement:

   SOURCE,MAP,LIST,FLAGW, XREF (SOURCE and FLAGW are defaults)

2. Include a WITH DEBUGGING MODE clause in the SOURCE-COMPUTER paragraph.

3. Insert a RESET TRACE statement at the beginning of the Procedure Division (any TRACE statement, even if not executed, causes a paragraph/section to be built).

4. Insert debugging statements in the source program. Use either USE FOR DEBUGGING declarative procedures, debugging lines in the main body of the program, or a combination of both.

5. Compile the program.

6. If debugging declaratives were coded, the object-time debugging switch must be set to an on position when executing the object program.

7. After program checkout is complete, the WITH DEBUGGING MODE clause should be removed. The program can then be recompiled. All debugging declarative paragraphs and debugging lines are treated as comments.

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S .........IDENTFCN SEQ/NO S

     PROCESS LIST,MAP
   1 IDENTIFICATION DIVISION.
   2 PROGRAM-ID.  TESTER.
   3   AUTHOR. PROGRAMMER NAME.
   4   INSTALLATION.  ROCHESTER LABORATORY.
   5   DATE-WRITTEN.  MARCH 6, 1979.
   6   DATE-COMPILED.  10/01/81.
   7 ENVIRONMENT DIVISION.
   8 CONFIGURATION SECTION.
   9   SOURCE-COMPUTER.  IBM-S34.
  10   OBJECT-COMPUTER.  IBM-S34.
  11 INPUT-OUTPUT SECTION.
  12 FILE-CONTROL.
  13     SELECT FILE-1 ASSIGN TO DISK-SAMPLE.
  14     SELECT FILE-2 ASSIGN TO DISK-SAMPLE.
  15 DATA DIVISION.
  16 FILE SECTION.
  17 FD  FILE-1
         LABEL RECORDS ARE STANDARD
         BLOCK CONTAINS 5 RECORDS
         RECORD CONTAINS 20 CHARACTERS
         DATA RECORD IS RECORD-1.
  18   01 RECORD-1.
  19     02 FIELD-A PICTURE IS X(20).
  20 FD  FILE-2
         LABEL RECORDS ARE STANDARD
         BLOCK CONTAINS 5 RECORDS
         RECORD CONTAINS 20 CHARACTERS
         DATA RECORD IS RECORD-2.
  21   01 RECORD-2.
  22     02 FIELD-A PICTURE IS X(20).
  23 WORKING-STORAGE SECTION.
  24   01 FILLAR.
  25     02 KOUNT PIC S99 COMP.
  26     02 ALPHABET PICTURE X(26) VALUE 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.
  27     02 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.
  28     02 NUMBR PIC S99 COMP.
  29     02 DEPENDENTS PIC X(26) VALUE '01234012340123401234012340'.
  30     02 DEPEND REDEFINES DEPENDENTS PICTURE X OCCURS 26 TIMES.
  31   01 WORK-RECORD.
  32     02 NAME-FIELD  PICTURE X.
  33     02 FILLER PICTURE X VALUE IS SPACE.
  34     02 RECORD-NO PICTURE 9999.
  35     02 FILLER PICTURE X VALUE IS SPACE.
  36     02 LOCATION PICTURE AAA VALUE IS 'NYC'.
  37     02 FILLER PICTURE X VALUE IS SPACE.
  38     02 NO-OF-DEPENDENTS PICTURE XX.
  39     02 FILLER PICTURE X(7) VALUE IS SPACES.
  40   01 RECORDA.
  41     02 A PICTURE S9(4) VALUE 1234.
```

Figure 9-13 (Part 1 of 2). Sample Source Listing

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S .........IDENTFCN SEQ/NO S
42       02 B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.                    G
43   PROCEDURE DIVISION.
44   BEGIN. READY TRACE.
     *     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
     *     AND INITIALIZES COUNTERS.
46   STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO KOUNT NUMBR.
     *     NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
     *     CONTAINED IN THE FILE, WRITES THEM ON DISK, AND DISPLAYS
     *     THEM ON THE CONSOLE.
49   STEP-2. ADD 1 TO KOUNT, ADD 1 TO NUMBR, MOVE ALPHA (KOUNT) TO
          NAME-FIELD.
53         COMPUTE B = B + 1.
54        MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.
55        MOVE NUMBR TO RECORD-NO.
56   STEP-3. DISPLAY WORK-RECORD.                  WRITE RECORD-1 FROM
          WORK-RECORD.
59   STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
     *     NOTE THAT THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS
     *     INPUT.
61   STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
     *     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES OUT
     *     EMPLOYEES WITH NO DEPENDENTS.
64   STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
67   STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO '0' MOVE 'Z' TO
70        NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO
          STEP-6.
72   STEP-8. CLOSE FILE-2.
74        RESET TRACE.
75   STEP-9.                                                               B
76   STEP-10.  GO TO STEP-9.
78        STOP RUN.
```

Figure 9-13 (Part 2 of 2). Sample Source Listing

DATA DIVISION MAP (F)

	STNO	ADDR	LNTH	NAME	STNO	ADDR	LNTH	NAME
F	17	009E	20	FILE-1 (K)				
D	18	0060	20	RECORD-1	19	0060	20	FIELD-A
F	20	032D	20	FILE-2				
D	21	0078	20	RECORD-2	22	0078	20	FIELD-A
D	24	000C	56	FILLAR	25	000C	2	KOUNT
	26	000E	26	ALPHABET	27	000E	1	ALPHA
	28	0028	2	NUMBR	29	002A	26	DEPENDENTS
	30	002A	1	DEPEND				
D	31	0044	20	WORK-RECORD	32	0044	1	NAME-FIELD
	34	0046	4	RECORD-NO	36	004B	3	LOCATION
	38	004F	2	NO-OF-DEPENDENTS				
D	40	0058	4	RECORDA	41	0058	4	A
	42	0058	4	B (H)				

Figure 9-14. Sample Data Division Map

```
                                PROCEDURE DIVISION MAP

PROCEDURE NAME                  STNO ADDR   STNO ADDR   STNO ADDR   STNO ADDR   STNO ADDR

PROCEDURE DIVISION PROLOGUE       43 05B1
  BEGIN                           44 0613     45 0619
  STEP-1                          46 061D     47 0623     48 062D
  STEP-2                          49 0639     50 063F     51 0645     52 064B     53 0656
                                  54 066D     55 067C
  STEP-3                          56 0686     57 068C     58 0698
  STEP-4                          59 06B0     60 06B6
  STEP-5                          61 06D9     62 06DF     63 06E7
  STEP-6                          64 06F1     65 06F7     66 070D
  STEP-7                          67 0711     68 0717     69 0724     70 0732     71 0743
  STEP-8                          72 0747     73 074D     74 0755
  STEP-9                          75 0759
  STEP-10                         76 075F     77 0765     78 0769

  PROGRAM SIZE = DATA DIVISION + PROCEDURE DIVISION + LITERALS + DTF/BUFFERS

      2168             140              448              50          1530

                                DIAGNOSTICS

 ERROR  LVL STNO TYPE  TEXT

CBL-0436 W    53  P    HIGH-ORDER TRUNCATION MAY OCCUR

  0 E LEVEL MESSAGES      0 C LEVEL MESSAGES      1 W LEVEL MESSAGES

END OF COMPILATION
```

Figure 9-15. Sample Procedure Division Map

```
                OVERLAY LINKAGE EDITOR STORAGE USAGE MAP AND CROSS REFERENCE LIST

 START      OVERLAY   CATEGORY  NAME AND       CODE LENGTH        REFERENCED BY
ADDRESS   NUMBER AREA            ENTRY     HEXADECIMAL DECIMAL

0000                      128     TESTER  (A)      0878    2168
0000                              TESTER
0878                        0     @CB000  (I)      0059      89   TESTER @CB590
0885                              @CB001                         @CB590
08D1                        0     @CB010           00AE     174   TESTER
08E2                              @CB012                         TESTER
08DE                              @CB011
097F                        0     @CB060           010C     268   TESTER @CB050
0A11                              @CB061                         @CB050
0A8B                        0     @CB070           007E     126   TESTER @CB050
0AB3                              @CB071                         @CB050
0B09                        0     @CB300           00DA     218   TESTER
0B20                              @CB301                         TESTER
0B19                              @CB302
0BE3                        0     @CB350           0053      83   TESTER
0BE3                              @CB350                         TESTER
0C36                        0     @CB400           012A     298   TESTER
0D60                        0     @CB410           004A      74   TESTER
0DAA                        0     @CB420           022E     558   TESTER @CB590 @CB510
0DBA                              @CB421                         TESTER @CB590 @CB510
0DC1                              @CB422
0DC8                              @CB423
0DCF                              @CB424                         @CB590
0DD6                              @CB425                         TESTER @CB590 @CB510
0FD8                        0     @CB510           01C6     454   TESTER
10E8                              @CB511                         TESTER
111E                              @CBSTK  (C)                    @CB600
119E                        0     @CB520           000D      13   @CB420 TESTER @CB600
11AB                        0     @CB580           026D     621   TESTER
11B4                              @CB581
1418                        0     @CB590           00F3     243   TESTER
150B                        0     @CB610           0067     103   @CB300 TESTER
1572                        0     @CB620           0141     321   @CB580 @CB510 @CB420 TESTER @CB150
                                                                  @CB600
1665                              @CB621                         TESTER
1659                              @CBSAV                         @CB580 @CB150
1690                              @CB629                         @CB510 @CB420 @CB600
16B3                        0     @CB050           032A     810   @CB070 @CB060 @CB010 @CB000
16B9                              @CBTBL                         @CB070 @CB060 @CB010 @CB000
16CC                              @CBXR1                         @CB010 @CB000
16E6                              @CBZ31
1799                              @CBGPM                         @CB010 @CB000
17EC                              @CB051
1802                              @CBPOP                         @CB010 @CB000
191F                              @CBXIT                         @CB070 @CB060 @CB010
192C                              @CBGOP                         @CB010 @CB000
19DD                        0     @CB340           0210     528   @CB410 @CB400 @CB300
1BED                        0     @CB600           013E     318   @CB340 @CB620 @CB580 @CB400 @CB350
                                                                  @CB300
1D2B                        0     @CB150           00C4     196   @CB590 @CB420
1D70                              @CBVL1                         @CB590 @CB420
1D77                              @CBVL2
1D42                              @CBNUM                         @CB590
1D32                              @CBIX1                         @CB590 @CB420
1D34                              @CBIX2                         @CB420
1D36                              @CBARR                         @CB590 @CB420
1D38                              @CBLEN                         @CB590 @CB420
1D3A                              @CBADR                         @CB590
1D3C                              @CBADL                         @CB420
1D41                              @CBNDD                         @CB590


SYS-3130 I TESTER MODULE'S MAIN STORAGE SIZE IS
           7663 DECIMAL
SYS-3131 I 0000 IS THE START CONTROL ADDRESS OF THIS MODULE
SYS-3134 I TESTER MODULE IS CATALOGED AS A LOAD MEMBER
           $LIBRARY IS THE LIBRARY NAME
               33 TOTAL NUMBER OF LIBRARY SECTORS
```

Figure 9-16 (Part 1 of 4). Sample Overlay Linkage Editor Storage Usage Map, Cross Reference List, and Execution Output

```
CBL-3024 **STNO= 00046, PRECEDING WERE 00044
CBL-3024 **STNO= 00049, PRECEDING WERE 00046 00044
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00046 00044
A 0001 NYC 0
CBL-3024 **STNO= 00059, PRECEDING WERE 00056 00049 00046 00044
CBL-3024 **STNO= 00049, PRECEDING WERE 00059 00056 00049 00046 00044
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00059 00056 00049 00046 00044
B 0002 NYC 1
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00059 00056 00049 00046 00044
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00059 00056 00049 00046 00044
C 0003 NYC 2
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00059 00056 00049 00046 00044
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00059 00056 00049 00046 00044
D 0004 NYC 3
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00059 00056 00049 00046 00044
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00059 00056 00049 00046
E 0005 NYC 4
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00059 00056 00049
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00059 00056
F 0006 NYC 0
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00059
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
G 0007 NYC 1
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
H 0008 NYC 2
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
I 0009 NYC 3
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
J 0010 NYC 4
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
K 0011 NYC 0
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
L 0012 NYC 1
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
M 0013 NYC 2
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
N 0014 NYC 3
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
O 0015 NYC 4
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
P 0016 NYC 0
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
Q 0017 NYC 1
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
R 0018 NYC 2
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
S 0019 NYC 3
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
```

Figure 9-16 (Part 2 of 4). Sample Overlay Linkage Editor Storage Usage Map, Cross Reference List, and Execution Output

```
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
T 0020 NYC 4
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
U 0021 NYC 0
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
V 0022 NYC 1
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
W 0023 NYC 2
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
X 0024 NYC 3
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
Y 0025 NYC 4
CBL-3024 **STNO= 00049, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00056, PRECEDING WERE 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
Z 0026 NYC 0
CBL-3024 **STNO= 00061, PRECEDING WERE 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049 00056
CBL-3024 **STNO= 00064, PRECEDING WERE 00061 00056 00049 00056 00049 00056 00049 00056 00049 00056 00049
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00061 00056 00049 00056 00049 00056 00049 00056 00049 00056
WORK-RECORD = A 0001 NYC Z
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00061 00056 00049 00056 00049 00056 00049 00056 00049
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00061 00056 00049 00056 00049 00056 00049 00056
WORK-RECORD = B 0002 NYC 1
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00061 00056 00049 00056 00049 00056 00049
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00061 00056 00049 00056 00049 00056
WORK-RECORD = C 0003 NYC 2
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00061 00056 00049 00056 00049
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00061 00056 00049 00056
WORK-RECORD = D 0004 NYC 3
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00061 00056 00049
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00061 00056
WORK-RECORD = E 0005 NYC 4
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00061
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = F 0006 NYC Z
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = G 0007 NYC 1
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = H 0008 NYC 2
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = I 0009 NYC 3
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = J 0010 NYC 4
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = K 0011 NYC Z
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
```

Figure 9-16 (Part 3 of 4). Sample Overlay Linkage Editor Storage Usage Map, Cross Reference List, and Execution Output

```
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = L 0012 NYC 1
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = M 0013 NYC 2
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = N 0014 NYC 3
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = O 0015 NYC 4
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = P 0016 NYC Z
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = Q 0017 NYC 1
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = R 0018 NYC 2
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = S 0019 NYC 3
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = T 0020 NYC 4
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = U 0021 NYC Z
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = V 0022 NYC 1
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = W 0023 NYC 2
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = X 0024 NYC 3
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = Y 0025 NYC 4
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00067, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
WORK-RECORD = Z 0026 NYC Z
CBL-3024 **STNO= 00064, PRECEDING WERE 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067
CBL-3024 **STNO= 00072, PRECEDING WERE 00064 00067 00064 00067 00064 00067 00064 00067 00064 00067 00064
```

Figure 9-16 (Part 4 of 4). Sample Overlay Linkage Editor Storage Usage Map, Cross Reference List, and Execution Output

```
STORAGE DUMP                          SYSTEM

STORAGE DUMP HAS BEEN REQUESTED...

SSP     REL  09   MOD  00

MCODE   REL  10   MOD  00    PATCH  0929

IAR-092B AR-08D3 X1-52A8 X2-0058 PR-8021

ERRTCB 52A8  ERRACE        CURTCB 52A8

CURXAM     MIC 0016 OP/Q F400 DATE 81/10/01

PROGRAM    TESTER     PROCEDURE

TRACE START/END/OLDEST     C900/E800/DC90
```

Figure 9-17 (Part 1 of 5). Sample Storage Dump

```
                    TASK STORAGE DUMP                TCB  52A8
                    IAR-092B AR-08D3 X1-52A8 X2-0058 PR-8021
ADDR    00       04       08       0C       10       14       18       1C
0000    35100005 05B1F1F0 F0F1F8F1 F2F6C1C2 C3C4C5C6 C7C8C9D1 D2D3D4D5 D6D7D8D9   *.....10018126ABCDEFGHIJKLMNOPQR*
0020    E2E3E4E5 E6E7E8E9 F2F6F0F1 F2F3F4F0 F1F2F3F4 F0F1F2F3 F4F0F1F2 F3F4E0F1   *STUVWXYZ260123401234012340123401*
0040    F2F3F4F0 E940F0F0 F2F640D5 E8C340E9 40404040 40404040 5253571F 00000000   *2340Z 0026 NYC Z        ........*
0060    E940F0F0 F2F640D5 E8C340F0 40404040 40404040 00000000 E940F0F0 F2F640D5   *Z 0026 NYC 0        ....Z 0026 N*
0080    E8C340F0 40404040 40404040 F0F1F2F3 F4F50000 00000000 00600000 0000A000   *YC 0        012345.....-.....*
00A0    0000FFFF FFFF0060 40404400 02000014 E2C1D4D7 D3C54040 0120031F 00FA0000   *........-  .....SAMPLE  ........*
00C0    02000127 031F033A FF033B27 0000001A 5E300140 40000000 1A000001 FF00733B   *.................;..  ..........*
00E0    03000100 6B020000 08000002 00000001 FFFF0003 EB34F400 F4430040 20A200A0   *.....................4.4.. ....*
0100    01200100 00000000 000052A8 00000000 033B0103 3B0200FA 009E0000 00000000   *.....................................*
0120    40404040 40404040 00000000 00000000 00000000 00000000 00000000 00000000   *                                    *
0140    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   *....................................*
                                    DUPLICATE LINES THRU 0300
0320    00000000 000U0000 78000000 00A00000 00FFFFFF FF007840 B0480002 000014E2   *.....................  .......S*
0340    C1D4D7D3 C5404003 B005AF03 89000002 0003AF04 B7033AFF 033B2700 00001A5E   *AMPLE  .....................;*
0360    30204000 0000001A 000001FF 034000F4 000000F4 FF000000 03038900 000001FF   *.. ..........  .4...4.............*
0380    FF0003EB 34F600F7 78004020 A200A003 B0010000 00000000 0052AB00 00000003   *.....6.7.. .....................*
03A0    3B00033B 01038903 2D000000 00000000 40404040 D540F0F0 F1F440D5 E8C340F3   *.................    N 0014 NYC 3*
03C0    40404040 40404040 D640F0F0 F1F540D5 E8C340F4 40404040 40404040 D740F0F0   *         O 0015 NYC 4        P 00*
03E0    F1F640D5 E8C340F0 40404040 40404040 D840F0F0 F1F740D5 E8C340F1 40404040   *16 NYC 0        Q 0017 NYC 1    *
0400    40404040 D940F0F0 F1F840D5 E8C340F2 40404040 40404040 E240F0F0 F1F940D5   *    R 0018 NYC 2        S 0019 N*
0420    E8C340F3 40404040 40404040 E340F0F0 F2F040D5 E8C340F4 40404040 40404040   *YC 3        T 0020 NYC 4        *
0440    E440F0F0 F2F140D5 E8C340F0 40404040 40404040 E540F0F0 F2F240D5 E8C340F1   *U 0021 NYC 0        V 0022 NYC 1*
0460    40404040 40404040 E640F0F0 F2F340D5 E8C340F2 40404040 40404040 E740F0F0   *         W 0023 NYC 2        X 00*
0480    F2F440D5 E8C340F3 40404040 40404040 E840F0F0 F2F540D5 E8C340F4 40404040   *24 NYC 3        Y 0025 NYC 4    *
04A0    40404040 E940F0F0 F2F640D5 E8C340F0 40404040 40404040 00000000 00000000   *    Z 0026 NYC 0        ........*
04C0    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   *....................................*
                                    DUPLICATE LINES THRU 0580
05A0    00000000 00000000 00000000 00000000 00F28743 90E3C5E2 E3C5D900 00005400   *..............2...TESTER.....*
05C0    4D020000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   *(...............................*
05E0    000000FF 01FF8D00 0005B100 00000000 000000C0 87166534 0105BCC2 0105B474   *.........................B....*
0600    020A7408 0CC08715 72C08710 E805E2C0 870613C0 BF0FD800 2C3A0105 E2C0BF0F   *............Y.8.....Q.....8...*
0620    D8002EC0 BF0C3600 9E000004 FF040100 0D077704 01002907 77C0BF0F D8003106   *#Q.....................Q...*
0640    10000D07 7B061000 290778C0 BF11AB07 791C0000 44FFC0BF 08E20000 00077B00   *.....................8.....*
0660    9006005B C0BF087B 00980600 5BC0BF11 AB07831C 00004FFF 3C400050 04210049   *...$.......$.........|.. .&....*
0680    00293AF0 0049C0BF 0FD80038 C0BF0DBA 00130057 C0BF0DD6 0C130073 0057C0BF   *#...0.....Q.............O.......*
06A0    0B20009E 06ACC0BF 0BE3009E C0BF06B0 C0BF0FD8 003B0411 07FF078E 071107FF   *.........T.........Q.....8...*
06C0    000DF281 14C20106 D1340106 AFC0BF06 39340806 AFC0BF06 B6C0BF0F D8003DC0   *..2..B..J.................Q...*
06E0    BF0D6000 9E00FFC0 BF0C3603 2D000008 FFC0BF0F D80040C0 BF0B0903 2D0703C0   *..-.....................Q. .....*
0700    BF070D0C 13005700 8BC0BF07 11C0BF07 47C0BF0F D800430D 00004F07 8FF20104   *.....................Q.....|..2..*
0720    3D400050 C0010732 0C00004F 07903C40 0050C0BF 14180808 000A079B 00001300   *#. .&.......|... .&.............*
0740    570000C0 BF06F1C0 BF0FD800 48C0BF0D 60032D00 FF3B0105 E2C0BF0F D8004BC0   *.....1...Q.....-.......S...Q...*
0760    BF0FD800 4CC0BF07 59C0BF11 9EC0BF11 9EF41000 8400F0F0 F101FF0F 00010000   *#..Q.(.............4....001.....*
0780    01000D01 FF2B0001 00000100 0DF2F6F0 E9E6D6D9 D260D9C5 C3D6D9C4 00000000   *.............260ZWORK-RECORD....*
07A0    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   *....................................*
                                    DUPLICATE LINES THRU 07C0
07E0    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00F0F0F0   *.............................000*
0800    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   *....................................*
                                    DUPLICATE LINES THRU 0840
0860    00000000 00000000 00000000 00000000 00000000 00000000 F28706C3 C2F0F0F0   *.....................2..CB000*
0880    063C2C08 9C340116 CCC20116 B9740811 74040F74 0215C087 1799F287 00C08719   *.........B.............2.....*
08A0    2C787002 F2100978 4002F290 037AF02D C0871799 784007F2 900F5B11 1B347408   *....2... .2..:0..... .2..$.....*
08C0    3B5C011A 395F011A 3B3C0008 9CC08718 02F28706 C3C2F0F1 F0063C87 08FC3C80   *.*...¬.........2..CB010.......*
08E0    094B3401 16CCC201 16B97408 1174040F 740215C0 871799C0 87192CF2 80501C13   *.....B.................2.&..*
0900    097E2DC0 871799C0 87192C4C 134D097E 7D0F3AF2 041B5800 3E4D5800 1E2D5603   *.=.........(.(.='..2.....(.....*
0920    1E3E7CF0 3E561E4D 2D54E02D 3458002D 1E563F2D 4D5B111B 3474084F 5C011A39   *#..@0...(.............($.....|*...*
0940    5F011A4F F208037A 8016C087 18023C80 08FC3C87 094B7B07 0F56E42D 3474044D   *¬..|2..:.............@...U....(*
0960    7BFB4D5E 000F4DC0 87191F00 F0F0F0F0 F0F0F0F0 F0F0F0F0 F0F0F1F2 F0F0F0F0   *#@8(;..(.....0000000000000000012*
0980    8706C3C2 F0F6F006 340209FB C202097F 5B111B34 B4081E7B F02DC201 0000AF07   *#..CB060.....B..'$......@0.B.....*
09A0    9191F287 14AC0789 91AE0791 91AE0791 91AE0791 89AE0791 91AF0789 899B0389   *#..2.....................*
09C0    00AE0791 8979F000 D20101E0 9026C201 16B97802 3BF2900B BB0F89AF 078991AC   *.....0.K.....B......2.........*
09E0    079189BC 017A7D03 04F2040C BC037A7D 0804F204 03BC077A 2C000000 91C08719   *.....:'..2....:'..2....:.......*
0A00    1F000000 00000000 00000000 00000000 0034020A 3FC2020A 08B40882 AF0F0808   *.....................B..........*
0A20    BC0A68BC 01347D03 04F2040F BC03347D 0804F204 06BC0734 BC00688C 000B0000   *......'..2.....'..2.............*
0A40    AC004134 AC005C34 AE000808 F2200ABC 207DAF07 0008F287 07AC0700 08BC007D   *......#.....2....'....2.......'*
0A60    BA0100AE 000000F2 811A5608 2D2DF287 00560924 24F20803 7A0124E0 205B7A01   *.......2......2.....2.:....$:.*
0A80    2DE0875B 7B002DC0 870000F2 8706C3C2 F0F7F006 0C010AF5 0AF70C05 0AE70B08   *#...$@.....2..CB070....5.7...X..*
0AA0    BC0F009B 01002D7B 023BF290 18BB0200 F2871234 080AF50C 050AE70B 026B012D   *..........2.....2.....5...X.....*
0AC0    0068022D 003C010A FC1E000A FC04D201 2D0E000A FC0AF9F2 82143601 0AF93602   *..............K.......92.....9..*
0AE0    0AFB9803 00019B01 0000C087 0AD1C201 16B9C087 191F191F FFFEFFFF FF6B0301   *.............JB.............*
0B00    00680298 03000198 01F28706 C3C2F3F0 F0083C80 0B53F287 0B3C200B 53F28704   *.........2..CB300.....2......2..*
0B20    3C400B53 3C040BD4 3C080BE0 34080B33 C20106FB 75020134 020B9DBD FF00F201   *. .....M.........B.............2.*
0B40    06BC420A F2875136 020BE2AC 010F0175 0201BC80 0BB8010D F2907EB8 410CF290   *...2.....8.............2.=...2.*
0B60    09B8400B F2102DF2 876FB802 0CF2900F B8200BF2 101EB880 0BF21018 F2875AB8   *.. .2..2.?...2.....2.....2..2.!.*
```

Figure 9-17 (Part 2 of 5). Sample Storage Dump

```
                        TASK STORAGE DUMP                TCB  52A8
                        IAR-092B AR-08D3 X1-52A8 X2-0058 PR-8021
ADDR    00       04       08       0C       10       14       18       1C
0B80    040CF290 09B8400B F21009F2 874BB880 0BF29045 F4010413 C08719DD 032DBD40   *..2... .2..2.....2..4..... *
0BA0    0AF2010A B8B00BC0 10150B75 1003BD70 0AF2011B BD440CF2 01153602 0BE2AD01   *.2.............2.....2.....S..*
0BC0    0355E202 06F20107 3C060BE0 F2870ABD 410AD001 043C030B E0C0871B ED0B3300   *..S..2......2.................*
0BE0    08FFFAF2 8706C3C2 F3F5F007 340B0BFB 34010C33 34020C35 C2010000 75020136   *...2..CB350.............B.....*
0C00    020C318D 01000C2F C0810C24 B50200E0 87003501 0C333502 0C350E01 0BFB0C2D   *.....................................*
0C20    35100BFB C0871BED 0BFB0035 00020000 FFFF0000 0000F287 06C3C2F4 F0F00834   *.....................2..CB400..*
0C40    080C46C2 0106EB75 02013402 0D0B3602 0D5B9C01 0A037B10 04F29003 BA400075   *...B.............$.......2... ..*
0C60    0201B801 0DF2900B C0871BED 0C460016 BD0000F2 0108C087 1BED0C46 0012BDC0   *.....2.............2.....2.....*
0C80    00F2B16F B0E000F2 B169B880 0CF29024 781004F2 1006B820 0DF2101B B8200DBA   *.2.?...2.....2.....2.....2.....*
0CA0    100D7905 04F2100C BB100D78 1004F290 03BA200D 782804F2 902B2C00 0D5E0CBC   *.....2.......2.......2.....;..*
0CC0    F00CF401 0402BD45 0AF2010E BC400ABC FF008C00 0C0D5EF2 8726BC40 0A8C000C   *0.4.....2... .....;2... ....*
0CE0    0D5EF287 0A7B4104 F2900498 000C0498 030C048C 01070D55 F401040C F4010402   *.;2.....2.............4...4...*
0D00    BC000BC0 8719DD03 2DBDC000 C0810000 BDE000F2 0131BC01 1AF40104 1336020D   *.................2.....4......*
0D20    5D2C000D 2C022E00 0D2C02AF 000101BC F0012C00 0D3E020F 000D3E0D 5FAC0000   *).............0.........¬...*
0D40    01BCF101 F28700D2 01057DFF 00C0010C 47D08701 FFFFFFFB FFF3FFF5 FFE30002   *..1.2..K..'.............3.5.T..*
0D60    F28706C3 C2F4F1F0 0734080D 70C20107 51750201 34020D8F B8010DF2 901B8C01   *2..CB410.....B.........2.....*
0D80    070DA7F4 010403BC 400AC087 19DD032D 7D4002F2 0103BC00 00D20103 7DFF00C0   *...4.... .....' .2.....K..'...*
0DA0    010D71D0 8701FFFF 0001F287 06C3C2F4 F2F0073C 000DECF2 87203C4D 0DECF287   *..........2..CB420.....2...(..2.*
0DC0    193C560D ECF28712 3C5C0DEC F2870B3C 790DECF2 87043CC9 0DEC3408 1D363401   *.....2...*..2......2....I.....*
0DE0    1D323402 1D34C201 0E74F287 C9C0871D 704C019A 1D3C3D4A 1D38F204 141F011D   *......B...2.I...(.....¢..2.....*
0E00    387F5C00 987FD202 95F40104 055E019A 7F4C0098 1D385E00 9881D202 95F40104   *.."*.."K..4...;..."(...;...K..4..*
0E20    05C20216 90F40004 0FD2029D F4010405 B8200AC0 10119EF2 87837C0A 5A7C7787   *.B...4....K..4.........2..@.!@..*
0E40    F2870F7A 40AAF287 037B40AA 7C005A7C 4A870CB0 0FD60FD7 7B07947B 07AF5C01   *2..! .2..@ .@.!@¢....O.P@..@..*.*
0E60    8D945C01 8587C087 1D704C01 891D384C 018F1D3C 5C018B89 5D018B85 F204045C   *..*.......(....(....*...)...2..*#
0E80    018B855E 018D8B5E 018F885C 04208F0C 6A0F9211 885E018D 815F0185 885F0185   *...;...;...*....!...;...¬...¬..*
0EA0    81D08255 5E018F81 5F01898B 5F018981 D00200F2 87075D01 8587D082 5535011D   *....;...¬...¬......2..).......*
0EC0    3235021D 3435101D 3674087D F2870AD2 02AAF401 0405F287 07D20290 F4010406   *...........'2..K..4...2..K..4...*
0EE0    0CB00FD6 0FD75C01 8D945C01 8587C087 0E8D004B 00010004 00770077 FFFF006A   *...O.P*...*.....................|*
0F00    0F281189 4001780F 28430001 00000000 00DB0010 0CC3C2F4 F230170B 00004300   *.... ..............CB42.......*
0F20    014B0F28 00004040 40404040 40404040 40404040 40404040 40404040 40404040   *......                         *
0F40    40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040   *                               *
                                  DUPLICATE LINES THRU 0FA0
0FC0    40404040 40404040 40404040 40404040 40404040 40404040 F28706C3 C2F5F1F0   *                           2..CB510*
0FE0    0634080F E8C20107 5DC20210 F99C011C 01A70411 11E2011A 2F011115 00F28209   *....YB..)B..9.......S.......2..*
1000    1604110A 00C0870F F82E0111 1500E202 02360111 03BDFF00 C0010FF8 3D00111B   *.........B.....S..........8....*
1020    C00110B5 3CFF111B C2021690 F400040F C202118F F4010409 C202111E BD7B00C0   *.........B...4...B...4...B....@..*
1040    811051BD FF02C081 105CE202 01C08710 3CE20204 8C040011 88F28739 0C6A1188   *.........*8......S.......2...!..*
1060    11893C7B 11263C7B 113C3C16 105BC087 10383402 10C03402 10C43402 118C0F01   *...@...@.....$...........D......*
1080    118C118E 0F01118C 11053C39 105BE202 01C08710 4A340210 BE340210 BA360211   *.............$8.....¢...........*
10A0    1A340210 B80E0410 BA111A0D 0110B811 1DC08210 A50C3B11 85117F0C 05114911   *.....................".....*
10C0    330C0411 33110A3D 0005E2F2 810FC087 0DBA006A 1188C087 0DD64F287 043C6A10   *..........S2.......!.....02...!.*
10E0    D335010F E8D08702 340810EF C201060D 1C0110CA 01D08702 271003E8 00640000A   *L...Y.......B.............Y....*
1100    0001FFFF 0001F0F0 F0F7F0F0 F0F0F0F1 F0F0F0F0 FFFB0600 060006FF 1183C3C2   *......00070000010000.........CB*
1120    D360F3F0 F2F4405C 5CE2E3D5 D67E40F0 F0F0F7F6 6B40D7D9 C5C3C5C4 C9D5C740   *L-3024 **STNO= 00076, PRECEDING *
1140    E6C5D9C5 40F0F0F0 F7F540F0 F0F0F7F6 40F0F0F0 F7F540F0 40F0F0F0 F0F0F7F6   *WERE 00075 00076 00075 00076 000*
1160    F7F540F0 F0F0F7F6 40F0F0F0 F7F540F0 F0F0F7F6 40F0F0F0 F7F540F0 F0F0F7F6   *75 00076 00075 00076 00075 00076*
1180    40F0F0F0 F7F54040 4040FF00 15111D10 04302411 1E6BFFFF FFFF2C00 0000F287   * 00075    .....,.....2.*
11A0    06C3C2F5 F2F006F4 000404F2 870AC3C2 F5F8F007 3A801350 34011335 C2011201   *.CB520.4...2..CB580....&....B...*
11C0    7402B074 08B87502 B8B50201 5E01B8DB 6C02C302 E202033C 10120A5F 01D1D14C   *.............;...%.C.S.....¬.JJ(*
11E0    01CC135E 7880C1F2 900D2C01 11F3014E 01C30000 E2020279 03C1F210 D65F01BA   *...;..A2.....3.+.C..S....A2.O¬..*
1200    BA6C06CA 065C01D3 C57820C1 F2900D2C 01121808 4E01CA00 00E20202 E2020738   *.%...*.LE..A2.......+....S..S...*
1220    801350F2 90046C01 CC010E00 120A120A 7830C7F2 90045F01 CAD91C01 1243CA4C   *..&2..%..........G2..¬..R.....(*
1240    04BF000D 7880C7C0 9013BF78 90C7C010 13AF5F01 BFD97C08 C83C0012 6F7D00C4   *......G......G....¬..R@.H...?'.D*
1260    F281077C 10C83C01 126F5E01 BABA5E01 C5C5F220 045E01BA BF5F00C8 D9C00112   *2..@.H...?;...;.EE2..;...¬.HR...*
1280    6A7880C6 F290397B 80C65F00 C6D9F282 2F5C01CA BA5E01BA BA5E01BA BA5E01BA   *¦..F2..@.F¬.FR2..*...;...;...;..*
12A0    CA5E01BA BA5C01CA D35E01CA CA5E01CA CA5E01CA CA5E01BA D35E01BA CAD08709   *.;...*..L;...;...;...;..L;.....*
12C0    7880B9C0 1014015E 01D1BA5F 00C1D9C0 8711F735 02165938 801350F2 90097880   *.......;.J.¬.AR...7.......&2....*
12E0    C1F2907B F2872C5E 01C3D179 84C1F290 226C03D3 325C01CE C35E01D3 D55E01D1   *A2.@2..;.CJ..A2..%.L.*..C;.LN;.J*
1300    D55E01CE D55D01CE D1F282F5 5D01CED3 F284EE78 08C1F210 075C01B4 C3F28704   *N;..N)..J2.5)..L2....A2..*..C2..*
1320    5C01B0C3 7880CFC0 10135F7B 80CFC202 1103C201 FF44C087 06730019 F1F1F8F1   **..C.....¬*..B...B.........1181*
1340    D1D100FF 44000A00 00FF0002 00000043 00000000 8C00FF00 00000100 02000078   *JJ.............................*
1360    80D0C010 14105F01 CCD97C08 C85F01BA BA5E01BA BA5E00D3 D3F22004 5E01BACC   *......¬..R@.H¬..;...;.LL2.....*
13B0    5F00C8D9 C0011371 5F01D1BA 5D01D1D7 F2810678 80D0F290 773C3A13 A97808C1   *¬.HR...¬..J.).JP2.....2.......A*
13A0    F290043C 3C13A99C 0100D1C0 87132B58 02BBBD58 03BCBD58 02BDBE58 02BFBF57   *2.........J.....................*
13C0    40BFD9F2 823B7D04 C8F2040A 7C04C8F2 87045E01 BAC55700 BFD9C002 13D25F00   * .R2..'.H2..@.H2..;..E...R...K¬.*
13E0    C8D9D082 005C04C0 BF5E01C5 C55C01CA C55E01C5 C55E01C5 CAC08713 D3388013   *HR...*...;.EE*..E;.EE;.EE;.E....*
1400    D6388013 50F21008 C0871BED 13390001 C0871BED 13390027 F28706C3 C2F5F9F0   *O...&2.................2..CB590*
```

Figure 9-17 (Part 3 of 5). Sample Storage Dump

```
                        TASK STORAGE DUMP              TCB  52A8
                        IAR-092B AR-08D3 X1-52A8 X2-0058 PR-8021
ADDR    00       04       08       0C       10       14       18       1C
1420    0834011D 32C20214 9134081D 3635011D 369C0C6E 0C7A0800 D2010DB4 0161AE01   *.....B.............).!..K..../..*
1440    616EB840 62F2900C 2E011D36 72C0871D 42F2870E 2E011D36 74C0871D 708C0159   */>. .2.........2.............*
1460    1D3A8C01 571D38B8 8062F290 35AF016E 708C0001 1D382D01 1D386EF2 0404AC00   *.........2...>.........>2....*
1480    016E8C01 041D3AAC 000201B9 0862F210 084D0000 0000F281 56AC0310 044C0000   *.>.............2..<....2......<..*
14A0    0000AC02 1C67C087 0DBA000A 079BC087 0DCF0002 1508B840 62F29027 8C04481D   *.....<.....  ..... .2......*
14C0    41AC014D 64AC004B 46AC0079 4BAE014D 79C08708 85000000 00000008 000000AC   *...(...........(..............*
14E0    01594DC0 870DCF00 130057C0 870DD6C0 87074308 08000A07 9B000013 00570000   *..(...........D..............*
1500    00010006 00074D7E 400000F2 8706C3C2 F6F1F006 3408156B 34011563 34021567   *...... = ..2..CB610...,.....*
1520    2C01156D 110F0115 6D1571B5 01093602 156FB502 FB360115 6D360215 6D0F0015   *...._...._........?......_..._...*
1540    6C1571F2 82109CFF 00003601 156F3602 156FC087 153D0C00 155D156D 9C130000   *%..2.........?...?.......)._....*
1560    C20106FB C202032D C0870BAB FF13FF00 0001F287 53C3C2F6 F2F00640 F5F7F2F6   *B...B.............2..CB620. 5726*
1580    60C3C2F1 40C3D6D7 E8D9C9C7 C8E340C9 C2D440C3 D6D9D740 F1F9F7F9 40D3C9C3   *-CB1 COPYRIGHT IBM CORP 1979 LIC*
15A0    C5D5E2C5 C44JD4C1 E3C5D9C9 C1D34060 40D7D9D6 D7C5D9E3 E840D6C6 40C9C2D4   *ENSED MATERIAL - PROPERTY OF IBM*
15C0    F0F761F2 F661F7F8 34081682 4C013416 59340116 59360116 86340116 88D20103   *07/26/78....<.................K..*
15E0    4D013016 8AF2012B 3A08169D C2021694 F4010401 BD0010F2 0108C202 16A6F401   *.(....2......B...4......2..B...4.*
1600    0405C202 1690F400 040F7D02 0DF2844D F2876C7B 80001C00 16290D0E 00162916   *..B...4...'..2.(2.X@..........*
1620    8D75010C 7501014D 0100168C F2012E0C 00168416 290F0016 84168F36 01168434   *.*.......(....2...............*
1640    0116550C 00165216 84350116 59360116 844C000E 0000C201 05B4F287 22C0871B   *.................<....B...2.....*
1660    ED168200 14350116 591C0116 59341C01 16820C0E 01168216 8F75020A 750108C0   *.............................*
1680    87060900 00FFFD05 B1FF01FF FF010002 80071694 0059BF00 000002E0 00008408   *.............................*
16A0    00000800 10B8DB00 2001C3C2 F6F21183 010000C3 C2F0F5F0 08000098 0006005B   *..........CB62.....CB050.......$*
16C0    000000FF FFFFFF04 22066DFF 28110300 F1F10006 F0F0F0F0 F0F0F0F0 F0F0F0F0   *..........._.....11..0000000000000*
16E0    F5F2F5F3 F5F7F1FF FFF0F0F0 F0F0F100 1416E604 04F0F0F0 F0F0F0F0 F0F0F0F0   *5253571..000001...W..000000000000*
1700    F0F5F2F5 F3F5F7F1 E07408DF 5C00B21A 54F34C34 58004D2D 7AF02D78 8001F210   *05253571....*....3<...(.!0....2.*
1720    277CB4AF 7C2E745F 0074017D F5007880 07F29203 7C80BA5F 00B2017C 001AF282   *.@..@..¬...'5....2..@..¬...@...2.*
1740    2D5C001A B2F2870A 7B800017C B3AF5E00 1A017D12 1AF20408 5F00B21A 5E00B29A   *.*...2..#..@..;...'..2..¬...;...*
1760    7C2DB47C 4CB35F00 B4015C00 4C2D5C12 2D4CF287 1C7C00BA 56F22D35 5B111B34   *@..@(.¬...*.(.*..<2..@...2..$...*
1780    74084F5C 011A395F 011A4FF2 08037C80 1658002D 4DC08700 00340817 EB750211   *..|*...¬..|2..@.....(.........*
17A0    5F090909 6C010201 E2020278 7002F210 346C0206 02E20203 78FE05F2 90177BFE   *¬...X...S.....2..%...S....2..@.*
17C0    055E0005 053C1317 D41E0017 D4057C00 055E0106 00780C02 F2900A6C 0209027B   *.;......M...M.@..;.....2..%...@*
17E0    0402E202 03740211 C087089A 340816CA 340116C6 350116C4 0C0116C4 16C63510   *..S.............F...D...D.F..*
1800    16CA7808 02F2101F 5E001616 F220037A 080F7804 02C09019 1F75020D 5F010D37   *.....2..;...2..!.............¬...*
1820    3C13191C F287F40D 0116C416 C6C08118 370C0116 C616C47D 0001C001 17095D00   *....2.4...D.F.......F.D'......).*
1840    1A04F204 037CB016 5E001616 F2200C78 4007F290 067A080F F287C478 4002F290   *..2..@..;...2... .2..!..2.D. .2.*
1860    037AF02D 56F22D34 74043B79 7002F210 037AF02D 75020679 8002F210 23793002   *.!0..2........2..!0.......2.....*
1880    C0100000 1C001892 045C124E 2D54F32D 345C062D 4E782002 C010097F C0870ABB   *.........*.+..3..*..+.......".....*
18A0    78200778 013BF290 2076022F BC40011C 0018C304 782002F2 10081E00 18C32FF2   *......2.....  ....C....2.....C.2*
18C0    205DAC00 0001F287 561C0019 1C047930 02F21047 782002F2 10153C2D 18EB1F00   *.)....2.........2.....2........*
18E0    18EB0478 023BF290 327B2000 F2872C78 30023402 191AF290 0676022F F2870A1F   *......2..@..2.........2.....2...*
1900    01191A04 1E00191A 2F3C4E19 1878023B F290043C 6019183C 0000009C 00002D75   *..........+.....2...-...........*
1920    040F7502 15750113 351016CA 340819DC 787002F2 90197BF0 02F2907E 75020DE2   *...................2...0.2.=...S*
1940    02147402 0D6C122D 007C121A F2876B5C 001A0454 F32D3475 02067402 3B798002   *.....X...@..2.,*....3.........*
1960    F290461C 00197204 783002F2 90037602 2F6C002D 00793002 F2103F78 3002F210   *2.........2.....%.....2.....2.*
1980    085E013B 2F5F013B 0475023B 782002F2 100B6800 2D0156F2 2D34F287 1DBD6000   *.;...¬.........2.......2..2...-.*
19A0    F201177B 202DF287 11782002 F29007C0 870A11F2 8704C087 0AB37D00 01C00117   *2..@..2.....2......2......'.....*
19C0    09780102 F290127C F0175800 182D5700 17185800 2D177AF0 1BC08709 0BF28706   *....2..@0.............!0.....2..*
19E0    C3C2F3F4 F0073401 1BB63402 1BBA3408 1BC63608 1BC23408 1BBE3501 1BC61C01   *CB340.............F...B.......F..*
1A00    1BC60135 011BC636 011BCA1C 011BCC00 0C011BBC 1A3F0D01 1BCC1BC8 C0811BB3   *.F....F.................H....*
1A20    35011BC6 7DA000F2 81187DFF 00F28112 7DE000C0 811B7E7D C000C081 1B7EC087   *...F'..2..'..2..'.....='.....=..*
1A40    1BB33502 1BCC3602 1BC07D40 0AF20109 8C01001B CEC0871B B37D410A F201098C   *..........' .2..........'..2...*
1A60    01001BDA C0871BB3 7D420AF2 011E7D00 0BF2010F 78100DF2 9009BC01 001BCEC0   *........'..2..'..2.....2........*
1A80    871BB38C 01001BD0 C0871BB3 7D430AF2 01098C01 001BE2C0 871BB37D 440AF201   *...........'..2......S...'..2..*
1AA0    098C0100 1BD6C087 1BB37D45 0AF20109 8C01001B DEC0871B B37D460A F201098C   *.....O....'..2..........'..2...*
1AC0    01001BD4 C0871BB3 7D470AF2 01098C01 001BE8C0 871BB37D 500AF201 098C0100   *.....M....'..2......Y....'&.2....*
1AE0    1BD2C087 1BB37D60 0AF20109 8C01001B D4C0871B B37D620A F201098C 01001BD2   *.K....'-.2......M....'..2......K*
1B00    C0871BB3 7D700AF2 01187840 0CF29009 8C01001B DCC0871B B38C0100 1BD8C087   *....'..2... .2.............Q..*
1B20    1BB37D75 0AF20109 8C01001B DEC0871B B37D990A C0011B41 8C01001B E2C0871B   *..'..2..........'.........S...*
1B40    B37D480A F201098C 01001BD6 C0871BB3 7D490AF2 01098C01 001BDEC0 871BB37D   *.'..2......O....'..2..........'*
1B60    520AF201 098C0100 1BD6C087 1BB37D53 0AC0011B B38C0100 1BD4C087 1BB33502   *..2......O....'.........M.....*
1B80    1BCC3602 1BC07D40 0AF20109 8C01001B CEC0871B B37D410A F201098C 01001BDA   *......' .2..........'..2.......*
1BA0    C0871BB3 7D990AF2 01098C01 001BE2C0 871BB3C2 010751C2 02032DC0 870D9000   *....'..2......S....B...B........*
```

Figure 9-17 (Part 4 of 5). Sample Storage Dump

```
                        TASK STORAGE DUMP              TCB  52A8
                        IAR-092B AR-08D3 X1-52A8 X2-0056 PR-8021
ADDR    00       04       08       0C       10       14       18       1C
1BC0    01000200 03032D00 00FFFD00 00F0F0F1 F0F2F1F2 F2F2F3F2 F4F3F0F3 F4F9F0F9  #.............00102122232430349O9#
1BE0    F1F9F2F9 F3F9F4F9 F5F9F6F9 F7F28706 C3C2F6F0 F006C201 1C523408 1C09C202  #192939495969972..CB600.B.......B.#
1C00    1690F400 040FC202 00006C01 C6037D32 C5F28103 7C30C5B5 02016C01 4B00D202  #..4...B...X.F.'.E2..@.E...X...K.#
1C20    CAF40104 09D2024E 9D0303AA F2811CBD FF00F281 07E20201 C0871C28 7C40985C  #.4...K.+....2.....2..S......@ .*#
1C40    4997984C 15631CFC D202607C 00AB9C03 039D7DA0 4AF28207 5F004A49 BB300098  #...(....K.-@....../'.¢2..¬.¢.....#
1C60    02004A58 014A4A58 024A4B58 014B4BE2 02015E00 ABAC7D04 ABD00100 D202ADF4  #..¢..¢¢..¢.....S..;...'.....K..4#
1C80    01040057D 004CF281 07D202B5 F4010405 D202BDF4 010405C0 87119E90 0000111E  #...'.(2..K..4...K..4.............#
1CA0    40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040  #                                 #
                                    DUPLICATE LINES THRU 1CC0
1CE0    40404040 40404040 404040FF F0F0F0F0 C3C2D360 F3F0F1FB 407B7B7B 7B000143  #           .0000CBL-3018 @@@@...#
1D00    00024B1C A0000043 00024611 1E0000DB 00100CC3 C2F6F0FF FF030000 10043118  #...................CB60.........#
1D20    1CA04BFF FF000000 000000C3 F1F5F2D5 06110E11 0310DA00 6A118811 1E000000  #...........C152N.......!.......#
1D40    00003408 1D993401 1DE63501 1D361C02 1D3A043C 001D371C 021D3F02 C0871D9A  #.........W.....................#
1D60    0C011D41 1D3A0E01 1D361DEC C0871D96 3C001DCC F287043C 021DCC34 081D9934  #.....................2..........#
1D80    011DE635 011D361C 031D3A03 C0871D9A 0E011D36 1DEEC087 0E6A3408 1DEA3DFE  #..W.....................!......#
1DA0    1D39F282 163C001D 39F28409 0E011D3A 1D32F287 060E011D 3A1D3438 B01D37F2  #..2......2........2............2#
1DC0    90153B80 1D373501 1659D201 001E011D 383A1E01 1D3A3A0C 011D3C1D 3A0F011D  #.........K.....................#
1DE0    3C1D38C2 010E74C0 871D9000 05000403 02B0140C 02168080 80806B0C 02168080  #...B.....................;.....#
1E00    FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  #................................#
                                    DUPLICATE LINES THRU 1FE0
```

Figure 9-17 (Part 5 of 5). Sample Storage Dump

CHECKPOINT/RESTART FACILITIES

When a program is expected to run for an extended period of time, provision should be made for taking checkpoint information periodically during the run. A checkpoint is the recording of the status of a program and main storage. Thus, it provides a means of restarting a job at a prior checkpoint position rather than at the beginning, if for any reason processing is terminated before the normal end of the program. For example, some malfunction (such as a power failure) may occur and cause an interruption. Checkpoints are taken using the COBOL RERUN clause.

Restart is a means of resuming the execution of the program from a checkpoint rather than from the beginning. The ability to restart is provided by the System/34 SSP restart procedure; CRESTART.

The CHECKPOINT/RESTART facility is an optional System Support Program (SSP) addition. During a full system configuration, a program additions display contains the following prompt:

```
CHECKPOINT/RESTART? (0-NO, 1-YES)...0
```

The user must select the CHECKPOINT/RESTART facility if the COBOL Program Product is to be installed and the COBOL RERUN clause is to be used. For additional information on this facility, refer to the *System Support Reference Manual* and the *Concepts and Design Guide*.

RERUN Clause

The presence of the RERUN clause in the source program causes the checkpoint facility to be invoked. When the checkpoint facility is invoked, the following information is saved:

1. The display station work area (local data area, UPSI switch settings, print belt image or member name)
2. The entire nucleus
3. The user's region
4. The off-line multi-volume file control sector (if used)

Because the COBOL RERUN clause provides linkage to the checkpoint facility, any warnings and restrictions on the use of this program also apply to the use of the RERUN clause. Refer to the *System Support Reference Manual* and the *Concepts and Design Guide* for more information on restrictions.

Taking a Checkpoint

To take a checkpoint, the programmer must code at least one RERUN clause in the COBOL program. Checkpoint information is written in a user-specified area on the fixed disk. This area contains two alternating checkpoint records, with each checkpoint overlaying the oldest checkpoint taken. Thus, restart uses the last checkpoint taken. The user must not specify an OCL FILE statement when providing disk space for checkpoint records, because this file is allocated by the System/34 SSP.

In designing a program for which checkpoints will be taken, the programmer should consider that, upon restarting, the program must be able to continue as though it had just reached that point in the program at which the checkpoint occurred. It is the responsibility of the programmer to get the program from the point of restart to the point of termination. Therefore, the programmer should ensure that:

1. File handling will permit easy reconstruction of the status of the system as it existed at the time the checkpoint was taken. The user should avoid updating records directly in master files. For example, the application could be written such that an output transaction file is created and is used in a later (not checkpointed) run that performs the updates.

2. For files opened OUTPUT, all records written on the file at the time the checkpoint is taken should be unaltered at restart time. For files opened I-O, care must be taken to design the program so that a restart will not duplicate work that has been completed between checkpoint time and restart time. For example, suppose that checkpoint 5 is taken. By adding an amount representing the interest due, account XYZ is updated on a direct access file that was opened I-O. If the program is restarted from checkpoint 5 and if the interest is recalculated and again added to account XYZ, incorrect results will be produced.

3. If display stations are being used by the program, the display station must be acquired and/or opened and the proper format redisplayed. The programmer should be aware that multiple requestor terminal (MRT) programs are restarted as single requestor terminal (SRT) programs.

4. There is enough space on the fixed disk for the checkpoint record file. Refer to the *Installation and Modification Reference Manual* for information about the size of the checkpoint record file.

5. That no files have DISP-SHR.

Each time a checkpoint is taken, the checkpoint program issues an informational message to the console/work station operator. This message indicates that a checkpoint record was just saved and also gives the label of the checkpoint record file.

Restarting a Program

To restart a job from a checkpoint, there are a number of steps and restrictions that must be followed. Refer to the *System Support Reference Manual* and the *Concepts and Design Guide* for a complete descripton of the restart requirements.

INTERPRETING OUTPUT

The System/34 American National Standard COBOL compiler, COBOL object module, overlay linkage editor, and other system components can produce output in the form of printed listings; diagnostic and informative messages; and data files directed to printers, work stations, or mass storage devices. This section illustrates the format of this output and describes it. The same COBOL program is used for each example.

Compiler Output

The output of the compiler job step may include:

- A printed listing of the statements contained in the source program
- A cross reference of the Data Division and Procedure Division names
- A Data Division map containing a glossary of compiler-generated information
- A Procedure Division map containing the statement number and relative address of the first generated instruction for each verb
- An object program size based on compilation statistics
- Compiler diagnostic messages
- Diagnostic statistics
- An executable or nonexecutable object program

The presence or absence of various types of compiler output is determined by options specified on the PROCESS statement. The level of the printed diagnostic messages depends on the FLAGW or FLAGE option of the PROCESS statement. All output to be listed is written to the printer. Page ejection fo the source listing is done by placing the stroke (/) character in the continuation area of a comment line.

Figure 9-18 shows the compiler output produced by the sample program. Each type of output is numbered and each format within each type is lettered. The text that follows is an explanation of the figure:

1 *Compiler options.* The PROCESS statement, if specified, is printed immediately preceeding the Identification Division header.

2 *The source module listing.* The statements in the source program are listed exactly as submitted, with the exception of the sequence number coded in columns 1 through 6. These sequence numbers, if coded, appear to the right of the source statements. The source module is not listed if NOSOURCE is specified.

The following notations may appear on the source module listing:

C Denotes that the line was included in the source program with a COPY statement.

S Denotes that the line is out of sequence. Sequence checking is performed only if sequence numbers are coded in columns 1 through 6.

O Denotes that an error occurred in this or a previous line of a PROCESS statement.

A The compiler generated line number. The numbers are listed to the left of the source statements. This is the number referenced in diagnostic messages, the Data Division map, and the Procedure Division map. It is also the number printed as a result of the source language TRACE statement. Note that a statement number may span several lines, and that a line may contain more than one statement.

B *The programmer-specified sequence number.* Sequence numbers coded in columns 1 through 6 on the COBOL coding form appear to the right of the source statements on the listing.

3 *Cross Reference Listings.* The cross reference listing is produced when the XREF option is specified. The listing consists of separate parts for Data Division names and Procedure Division names. Each name is listed along with the statement numbers that contain references to the name. Statement numbers followed by an asterisk define references to the name.

4 *Data Division map.* The Data Divison map is listed when the MAP option is specified. The Data Division map contains information about names in the COBOL source program.

C *Descriptive code.* A descriptive code precedes specific data items. A list of these codes follows:

F–Identifies FD level file-names.
D–Identifies 01 level data-names.

D *The Statement number.* The compiler-generated statement number is listed for each data item in the Data Division map.

E *The relative address.* The relative address of each data item that appears in the Data Division map is listed in hex.

F *The length.* The decimal length of each data item appearing in the Data Division map is listed.

Note: No relative address or length is listed in the Data Division map, and no main storage is set aside for a nonnumeric level-88 item of length 1.

G *The name.* The data-name as specified in the source program is listed.

5 *The Procedure Division map.* The Procedure Division map is provided when the LIST option is specified. It associates a relative address with each compiler-generated statement number in the Procedure Division. A compiler-generated statement number is generated for each section-name, paragraph-name, and verb in the Procedure Division.

H *Compiler-generated statement number.* Note that some numbers appear in the Procedure Division map that do not appear on the source module listing. This is because there are cases where multiple verbs appear on the same line or cases where verbs appear on the same line as paragraph-names. Only the first generated number appears on each line of the source module listing, while every number appears in the Procedure Division map.

J *The address.* The relative address in hex for each verb, paragraph-name, or section-name is listed in the Procedure Division map.

6 *Compilation statistics.* The total size of the program, in decimal, is provided by the compiler. This size represents the sum of: the length of the Data Division, literals, internal data file buffers, and the amount of code generated by the compiler for Procedure Division statements.

7 *Diagnostic messages.* The final output of the compiler consists of the diagnostic messages generated for errors detected during compilation:

K *The statement number.* The compiler-generated statement number associated with the line of source code in which the error is detected is listed.

L *The error number.* A number is associated with each message issued by the compiler.

- *The severity level.* There are three severity levels as follows:

  W–Warning. This level indicates that a possible error was present in the source program. These warning messages are listed if the FLAGE option is not specified.

  C–Conditional. This level indicates that an error was made but the compiler usually makes a corrective assumption. The statement containing the error is retained. Execution can be attempted. These conditional messages are listed if the FLAGE option is not specified.

  E–Error. This level indicates that a serious error was made. Usually the compiler makes no corrective assumption. The statement containing the error is dropped. Compilation is completed, but execution of the program should not be attempted. If the FLAGE option is specified, only those diagnostics with a severity level of E are listed.

- *Type:*

  I–Identification or Environment Division or PROCESS statement diagnostic.

  D–Data Division diagnostic.

  P–Procedure Division diagnostic.

- *The message text.* The text identifies the condition that caused the error and indicates the action taken by the compiler.

- *Diagnostic statistics.* Lists the number of W, C, and E level messages, if any.

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S .........IDENTFCN SEQ/NO S (B)

 (A) PROCESS QUOTE,LIST,MAP,XREF [1]
  1  IDENTIFICATION DIVISION.                                              00010
  2  PROGRAM-ID.  TESTPR.                                                  00020
  3    AUTHOR. PROGRAMMER NAME.                                            00030
  4    INSTALLATION.  ROCHESTER LABORATORY.                                00040
  5    DATE-WRITTEN.  MARCH 6, 1979.                                       00050
  6    DATE-COMPILED.  79/05/01.                                           00060
  7  ENVIRONMENT DIVISION.                                                 00070
  8  CONFIGURATION SECTION.                                                00080
  9    SOURCE-COMPUTER.  IBM-S34.                                          00090
 10    OBJECT-COMPUTER.  IBM-S34.                                          00100
 11  SPECIAL-NAMES.                                                        00110
 12      SYSTEM-CONSOLE IS PRINTER.                                        00120
 13  INPUT-OUTPUT SECTION.                                                 00130
 14  FILE-CONTROL.                                                         00140
 15      SELECT FILE-1 ASSIGN TO DISK-SAMPLE.                              00150
 16      SELECT FILE-2 ASSIGN TO DISK-SAMPLE.                              00160
 17  DATA DIVISION.                                                        00170
 18  FILE SECTION.                                                         00180
 19  FD  FILE-1                                                            00190
         LABEL RECORDS ARE STANDARD                                        00200
         RECORD CONTAINS 20 CHARACTERS                                     00210
         DATA RECORD IS RECORD-1.                                          00220
 20    01 RECORD-1.                                                        00230
 21      02 FIELD-A PICTURE IS X(20).                                       00240
 22  FD  FILE-2                                                            00250
         LABEL RECORDS ARE STANDARD                                        00260
         RECORD CONTAINS 20 CHARACTERS                                     00270
         DATA RECORD IS RECORD-2.                                          00280
 23    01 RECORD-2.                                                        00290
 24      02 FIELD-A PICTURE IS X(20).                                       00300
 25  WORKING-STORAGE SECTION.                                              00310
 26    01 FILLER.                                                          00320
 27      05 KOUNT PIC S99 COMP-4.                                         00330
[2]28      05 ALPHABET PICTURE X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".   00340
 29      05 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.           00350
 30      05 NUMBR PIC S99 COMP-4.                                         00360
 31      05 DEPENDENTS PIC X(26) VALUE "01234012340123401234012340".      00370
 32      05 DEPEND REDEFINES DEPENDENTS PICTURE X OCCURS 26 TIMES.        00380
 33    01 WORK-RECORD.                                                    00390
 34      05 NAME-FIELD  PICTURE X.                                        00400
 35      05 FILLER PICTURE X VALUE IS SPACE.                              00410
 36      05 RECORD-NO PICTURE 9999.                                       00420
 37      05 FILLER PICTURE X VALUE IS SPACE.                              00430
 38      05 LOCATION PICTURE AAA VALUE IS "NYC".                          00440
 39      05 FILLER PICTURE X VALUE IS SPACE.                              00450
 40      05 NO-OF-DEPENDENTS PICTURE XX.                                  00460
 41      05 FILLER PICTURE X(7) VALUE IS SPACES.                          00470
 42  PROCEDURE DIVISION.                                                   00480
 43  BEGIN. READY TRACE.                                                   00490
    ************************************************************         00500
    *** THE FOLLOWING PARAGRAPH OPENS THE OUTPUT FILE TO BE CREATED***    00510
    ***   AND INITIALIZES COUNTERS.                              ***      00520
    ************************************************************         00530
 45  STEP-1. OPEN OUTPUT FILE-1.                                           00540
 47          MOVE ZERO TO KOUNT NUMBR.                                     00550
    ************************************************************         00560
    ***   THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE       ***    00570
    ***   CONTAINED IN THE FILE, WRITES THEM ON DISK, AND DISPLAYS ***    00580
    ***   THEM ON THE CONSOLE.                                   ***      00590
    ************************************************************         00600
 48  STEP-2. ADD 1 TO KOUNT. ADD 1 TO NUMBR.                               00610
 51      MOVE ALPHA (KOUNT) TO NAME-FIELD.                                 00620
 52      MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.                          00630
 53      MOVE NUMBR TO RECORD-NO.                                          00640
 54  STEP-3. DISPLAY WORK-RECORD.                                          00650
 56      WRITE RECORD-1 FROM WORK-RECORD.                                  00660
 57  STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.        00670
    ************************************************************         00680
    ***   THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS          ***      00690
    ***   INPUT.                                                 ***      00700
    ************************************************************         00710
```

Figure 9-18 (Part 1 of 4). Example of Compiler Output

```
59  STEP-5.                                                              00720
60      CLOSE FILE-1.                                                    00730
61      OPEN INPUT FILE-2.                                               00740
    ************************************************************       00750
    *** THE FOLLOWING READS BACK THE FILE AND SINGLES OUT EMPLOYEES***  00760
    *** WITH NO DEPENDENTS.                                      ***     00770
    ************************************************************       00780
62  STEP-6.                                                              00790
63      READ FILE-2 RECORD INTO WORK-RECORD                              00800
64          AT END GO TO STEP-8.                                         00810
65  STEP-7.                                                              00820
66      IF NO-OF-DEPENDENTS IS EQUAL TO "0"                              00830
67          MOVE "Z" TO NO-OF-DEPENDENTS.                                00840
68      EXHIBIT CHANGED WORK-RECORD.                                     00850
69      GO TO STEP-6.                                                    00860
70  STEP-8.                                                              00870
71      CLOSE FILE-2.                                                    00880
72      RESET TRACE.                                                     00890
73      STOP RUN.                                                        00900
```

3 CROSS REFERENCE

DATA NAME	REFERENCES					
ALPHA	29*	51				
ALPHABET	28*	29				
DEPEND	32*	52				
DEPENDENTS	31*	32				
FIELD-A	21*	24*				
FILE-1	15	19*	46	60		
FILE-2	16	22*	61	63	71	
KOUNT	27*	47	49	51	52	58
LOCATION	38*					
NAME-FIELD	34*	51				
NO-OF-DEPENDENTS	40*	52	66	67		
NUMBR	30*	47	50	53		
PRINTER	12*					
RECORD-NO	36*	53				
RECORD-1	19	20*	56			
RECORD-2	22	23*				
WORK-RECORD	33*	55	56	63	68	

CROSS REFERENCE

PROCEDURE NAME	REFERENCES	
BEGIN	43*	
STEP-1	45*	
STEP-2	48*	58
STEP-3	54*	58
STEP-4	57*	
STEP-5	59*	
STEP-6	62*	69
STEP-7	65*	
STEP-8	64	70*

Figure 9-18 (Part 2 of 4). Example of Compiler Output

4 DATA DIVISION MAP

C **D** **E** **F** **G**

```
   STNO ADDR LNTH NAME                                  STNO ADDR LNTH NAME

F    19 0092  20 FILE-1
D    20 0058  20 RECORD-1                                 21 0058  20 FIELD-A
F    22 0321  20 FILE-2
D    23 006C  20 RECORD-2                                 24 006C  20 FIELD-A
     27 000C   2 KOUNT                                    28 000E  26 ALPHABET
     29 000E   1 ALPHA                                    30 0028   2 NUMBR
     31 002A  26 DEPENDENTS                               32 002A   1 DEPEND
D    33 0044  20 WORK-RECORD                              34 0044   1 NAME-FIELD
     36 0046   4 RECORD-NO                                38 004B   3 LOCATION
     40 004F   2 NO-OF-DEPENDENTS
```

Figure 9-18 (Part 3 of 4). Example of Compiler Output

5 PROCEDURE DIVISION MAP

H **J**

```
PROCEDURE NAME                  STNO ADDR  STNO ADDR  STNO ADDR  STNO ADDR  STNO ADDR

PROCEDURE DIVISION PROLOGUE       42 05A5
  BEGIN                           43 0607    44 060D
  STEP-1                          45 0611    46 0617    47 0621
  STEP-2                          48 063D    49 0643    50 065F    51 067B    52 0686
                                  53 0695
  STEP-3                          54 06A3    55 06A9    56 06B5
  STEP-4                          57 06CD    58 06D3
  STEP-5                          59 06F8    60 06FE    61 0706
  STEP-6                          62 0710    63 0716    64 072C
  STEP-7                          65 0730    66 0736    67 0743    68 0751    69 0776
  STEP-8                          70 077A    71 0780    72 0788    73 078C
```

6

```
PROGRAM SIZE = DATA DIVISION + PROCEDURE DIVISION + LITERALS + DTF/BUFFERS

     2168            128                495              50          1495
```

7 DIAGNOSTICS

L **M** **K** **N** **O**

```
ERROR    LVL STNO TYPE   TEXT

CBL-0439 C     68   P    NAMED MISSING -- ASSUMED
```

P

```
0 E LEVEL MESSAGES      1 C LEVEL MESSAGES      0 W LEVEL MESSAGES

CBL-1019  C OR E LEVEL DIAGNOSTICS DETECTED
END OF COMPILATION
```

Figure 9-18 (Part 4 of 4). Example of Compiler Output

Linkage Editor Output

The output of the overlay linkage editor consists of:

- A map of the object program after it has been processed by the linkage editor. The map includes cross reference information.
- Linkage editor statistics.
- Any diagnostic messages associated with link editing.
- An executable object program which may be cataloged in the object library. The form that this object program takes, as well as whether or not it is to be retained, are options of the PROCESS statement.

Figure 9-19 is an example of a link editor output listing. The different types of output are numbered and each type to be explained is lettered.

1 *Overlay linkage editor map and cross reference list* The map and cross reference list are produced following the execution of the overlay linkage editor when the LIST option is specified on the compiler PROCESS statement:

- (A) *Start address* The main storage address, in hex, of the module or routine named in column B.
- (B) *Category* The category of the module, either as it is in the library, or via a category override. For a discussion of categories, refer to Program Linkage earlier in this chapter.
- (C) *Name* The name of the COBOL object module and its entry point, as well as the names and entry points of all object-time routines required to execute this object module.
- (D) *Code length* The length, in decimal and hex, is provided for the COBOL object module and object-time routines.
- (E) *Cross-reference lists* This list provides the names of all routines or object modules that refer to the routines named in column C.

2 *Linkage Editor statistics* The total main storage size required for the executable object program, the start address of this program, and library information for cataloging this program in the object library, are provided.

```
[1] OVERLAY LINKAGE EDITOR STORAGE USAGE MAP AND CROSS REFERENCE LIST
   (A)                    (B)         (C)              (D)                (E)
 START     OVERLAY    CATEGORY  NAME AND     CODE LENGTH       REFERENCED BY
ADDRESS  NUMBER AREA              ENTRY    HEXADECIMAL DECIMAL

 0000                    128    TESTPR        0878     2168
 0000                              TESTPR
 0878                      0    @CB000        0059       89    TESTPR @CB590
 0885                              @CB001                       TESTPR @CB590
 08D1                      0    @CB010        00AE      174    TESTPR
 08E2                              @CB012                       TESTPR
 08DE                              @CB011                       TESTPR
 097F                      0    @CB060        010C      268    TESTPR @CB050
 0A11                              @CB061                       @CB050
 0A8B                      0    @CB300        00A0      160    TESTPR
 0AA2                              @CB301                       TESTPR
 0A9B                              @CB302
 0B2B                      0    @CB350        0031       49    TESTPR
 0B2B                              @CB350                       TESTPR
 0B5C                      0    @CB400        010C      268    TESTPR
 0C68                      0    @CB410        0058       88    TESTPR
 0CC0                      0    @CB420        0206      518    TESTPR @CB590 @CB510
 0CD0                              @CB421                       TESTPR @CB590 @CB510
 0CD7                              @CB422
 0CDE                              @CB423
 0CE5                              @CB424                       @CB590
 0CEC                              @CB425                       TESTPR @CB590 @CB510
 0EC6                      0    @CB510        01C6      454    TESTPR
 0FD6                              @CB511                       TESTPR
 100C                              @CBSTK                       @CB600
 108C                      0    @CB520        000D       13    @CB420 TESTPR @CB600
 1099                      0    @CB580        0248      584    TESTPR
 10A2                              @CB581
 12E1                      0    @CB590        00E9      233    TESTPR
 13CA                      0    @CB610        0067      103    @CB300 TESTPR
 1431                      0    @CB620        0141      321    @CB580 @CB510 @CB420 TESTPR @CB150
                                                                @CB600
 1524                              @CB621                       TESTPR
 1518                              @CBSAV                       @CB580 @CB150
 154F                              @CB629                       @CB510 @CB420 @CB600
 1572                      0    @CB050        0324      804    @CB060 @CB010 @CB000
 1578                              @CBTBL                       @CB060 @CB010 @CB000
 158B                              @CBXR1                       @CB010 @CB000
 15A5                              @CBZ31
 1658                              @CBGPM                       @CB010 @CB000
 16AB                              @CB051
 16C1                              @CBPOP                       @CB010 @CB000
 17DE                              @CBXIT                       @CB060 @CB010
 17EB                              @CBGOP                       @CB010 @CB000
 1896                      0    @CB340        01E3      483    @CB410 @CB400 @CB300
 1A79                      0    @CB600        00E3      227    @CB340 @CB620 @CB580 @CB400 @CB300
 1B5C                      0    @CB150        00C4      196    @CB590 @CB420
 1BA1                              @CBVL1                       @CB590 @CB420
 1BA8                              @CBVL2

 START     OVERLAY    CATEGORY  NAME AND     CODE LENGTH       REFERENCED BY
ADDRESS  NUMBER AREA              ENTRY    HEXADECIMAL DECIMAL

 1B73                              @CBNUM                       @CB590
 1B63                              @CBIX1                       @CB590 @CB420
 1B65                              @CBIX2                       @CB420
 1B67                              @CBARR                       @CB590 @CB420
 1B69                              @CBLEN                       @CB590 @CB420
 1B6B                              @CBADR                       @CB590
 1B6D                              @CBADL                       @CB420
 1B72                              @CBNDD                       @CB590

[2] SYS-3130 I TESTPR MODULE'S MAIN STORAGE SIZE IS
                 7200 DECIMAL
    SYS-3131 I 0000 IS THE START CONTROL ADDRESS OF THIS MODULE
    SYS-3134 I TESTPR MODULE IS CATALOGED AS A LOAD MEMBER
                 #LIBRARY IS THE LIBRARY NAME
                     31 TOTAL NUMBER OF LIBRARY SECTORS
```

Figure 9-19. Example of a Linkage Editor Output Listing

COBOL Object Program–Execution Output

The output generated by program execution (in addition to data written on output files) can include:

- Data displayed on the console or on the printer
- Messages to the operator
- System informative messages
- System diagnostic messages

Figure 9-20 is an example of output from the execution job step.

1 *OCL statements.* The operation control language (OCL) statements for the execution job step are shown.

2 *Program output on printer.* The results of the execution of the TRACE statement, the DISPLAY statement, and the EXHIBIT NAMED CHANGED statement appear on the program listing. See *TRACE Statement* in Chapter 6 for further information on program output.

```
//  LOAD TESTPR          1
//  FILE NAME-SAMPLE, UNIT-F1, BLOCKS-10
//  RUN
**STNO= 00045, PRECEDING WERE 00043
**STNO= 00048, PRECEDING WERE 00045 00043
**STNO= 00054, PRECEDING WERE 00048 00045 00043
A 0001 NYC 0
**STNO= 00057, PRECEDING WERE 00054 00048 00045 00043
**STNO= 00048, PRECEDING WERE 00057 00054 00048 00045 00043
**STNO= 00054, PRECEDING WERE 00048 00057 00054 00048 00045 00043
B 0002 NYC 1
**STNO= 00048, PRECEDING WERE 00054 00048 00057 00054 00048 00045 00043
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00057 00054 00048 00045 00043          2
C 0003 NYC 2
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00057 00054 00048 00045 00043
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00057 00054 00048 00045 00043
D 0004 NYC 3
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00057 00054 00048 00045 00043
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00057 00054 00048 00045 00043
E 0005 NYC 4
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00057 00054 00048 00045 00043
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00057 00054 00048 00045
F 0006 NYC 0
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00057 00054 00048
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00057 00054
G 0007 NYC 1
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00057
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
H 0008 NYC 2
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
I 0009 NYC 3
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
J 0010 NYC 4
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
K 0011 NYC 0
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
L 0012 NYC 1
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
M 0013 NYC 2
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
N 0014 NYC 3
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
O 0015 NYC 4
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
P 0016 NYC 0
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
Q 0017 NYC 1
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
R 0018 NYC 2
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
S 0019 NYC 3
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
T 0020 NYC 4
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
U 0021 NYC 0
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
V 0022 NYC 1
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
W 0023 NYC 2
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
X 0024 NYC 3
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
Y 0025 NYC 4
**STNO= 00048, PRECEDING WERE 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00054, PRECEDING WERE 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
Z 0026 NYC 0
**STNO= 00059, PRECEDING WERE 00054 00048 00054 00043 00054 00048 00054 00048 00054 00048 00054 00048 00054
**STNO= 00062, PRECEDING WERE 00059 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
**STNO= 00065, PRECEDING WERE 00062 00059 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048 00054
```

Figure 9-20 (Part 1 of 2). Example of Output from the Execution Job Step

```
WORK-RECORD = A 0001 NYC Z
**STNO= 00062, PRECEDING WERE 00065 00062 00059 00054 00048 00054 00048 00054 00048 00054 00048 00054 00048
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00059 00054 00048 00054 00048 00054 00048 00054 00048 00054
WORK-RECORD = B 0002 NYC 1
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00059 00054 00048 00054 00048 00054 00048 00054 00048
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00059 00054 00048 00054 00048 00054 00048 00054
WORK-RECORD = C 0003 NYC 2
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00059 00054 00048 00054 00048 00054 00048
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00059 00054 00048 00054 00048 00054
WORK-RECORD = D 0004 NYC 3
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00059 00054 00048 00054 00048
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00059 00054 00048 00054
WORK-RECORD = E 0005 NYC 4
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00059 00054 00048
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00059 00054
WORK-RECORD = F 0006 NYC Z
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00059
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = G 0007 NYC 1
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = H 0008 NYC 2
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = I 0009 NYC 3
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = J 0010 NYC 4
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = K 0011 NYC Z
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = L 0012 NYC 1
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = M 0013 NYC 2
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = N 0014 NYC 3
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = O 0015 NYC 4
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = P 0016 NYC Z
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = Q 0017 NYC 1
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = R 0018 NYC 2
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = S 0019 NYC 3
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = T 0020 NYC 4
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = U 0021 NYC Z
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = V 0022 NYC 1
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = W 0023 NYC 2
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = X 0024 NYC 3
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = Y 0025 NYC 4
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00065, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
WORK-RECORD = Z 0026 NYC Z
**STNO= 00062, PRECEDING WERE 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065
**STNO= 00070, PRECEDING WERE 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062 00065 00062
```

Figure 9-20 (Part 2 of 2). Example of Output from the Execution Job Step

Diagnosed Source File

COBOL optionally builds a file that can be retrieved and displayed at a display station. Figure 9-21 illustrates a listing of this file. The content of the file is as follows:

- A $MAINT control statement (//COPY), which can be used to direct the file to a library type and member name. The library type is always S for source. The member name is the same as the LABEL parameter on the OCL FILE statement with the name $WORK2.
- One line of heading information, identifying the file as output from the COBOL compiler, with data and time.
- Two lines of compiler error summary information, containing the total number of errors and highest severity.
- Two lines of overlay linkage editor information, indicating the success or failure of the link step.
- COBOL source statements with compiler diagnostics embedded at the pertinent point, where possible.
- Additional compiler and link-editor diagnostics and informational messages. All statements which are merged with the COBOL source statements to create the diagnosed source file, including heading and summary records, have the following code in the first two positions of the record:

  ?C: indicates text added as a result of a COBOL COPY statement.
  ?R: indicates text added as a result of a COBOL COPY statement and text modified as a result of a REPLACING phrase in a COPY statement.
  ?I: indicates a compiler informational message.
  ??: indicates a linkage-editor informational message.

The user invokes this function at compile time, with a dsf parameter on the COBOL or COBOLCG command.

Example:

```
COBOL TESTPR,size,inlib,outlib,,,PRFILE
```

If the user is coding OCL instead of using the supplied procedures, another OCL file statement parameter is required for $WORK2, and an additional work file, $WORK3, must be defined.

Example:

```
// FILE NAME-$WORK2,LABEL-dsflabel...
// FILE NAME-$WORK3...
```

Upon completion of the compilation, the file named dsflabel is available for immediate examination through the use of the DISPLAY command (SYSLIST must be assigned to CRT). The user may prefer to examine and update the source program in one step through the use of SEU (Source Entry Utility). To accomplish this, the diagnosed source file must first be moved to a source library member. The COBMOVE command provides this function, or the user can enter either a TOLIBR command or the OCL and utility control statements for $MAINT. The SCAN feature of SEU makes it easy to find the inserted records.

After the source member has been updated to correct the identified errors, the source member can be submitted for recompilation. The compiler ignores all statements containing the insertion code in positions 1 and 2.

```
?I  IBM SYSTEM/34 ANSI COBOL PROGRAM PRODUCT
?I      6 E LEVEL MESSAGES.      0 C LEVEL MESSAGES      1 W LEVEL MESSAGES
?I
?I
?? 2924 DECIMAL IS THE MAIN STORAGE SIZE OF THE LOAD MEMBER
00001  PROCESS LINK,LET,MAP                                                  DSFTST
00002  IDENTIFICATION DIVISION.                                              DSFTST
00003  PROGRAM-ID. DSFTST.                                                   DSFTST
00004  ENVIRONMENT DIVISION.                                                 DSFTST
?E          CBL-0120 E   SOURCE-COMPUTER NOT IN CONFIGURATION SECTION
00005  SOURCE-COMPUTER. IBM-S34.                                             DSFTST
?E          CBL-0120 E   OBJECT-COMPUTER NOT IN CONFIGURATION SECTION
00006  OBJECT-COMPUTER. IBM-S34 MEMORY 10000 CHARACTERS.                     DSFTST
00007  INPUT-OUTPUT SECTION.                                                 DSFTST
?E          CBL-0161 W   EMPTY PARAGRAPH IN ENVIRONMENT DIVISION
00008  FILE-CONTROL.                                                         DSFTST
00009  DATA DIVISION.                                                        DSFTST
00010  FILE SECTION.                                                         DSFTST
00011  WORKING-STORAGE SECTION.                                              DSFTST
00012  77  NOM-KEY          PIC  X(14)    VALUE 'FFFFFFFFFFFFFF'.           DSFTST
00013  77  NEXT-KEY         PIC X(14).                                      DSFTST
?E          CBL-0207 E   INVALID LITERAL IN VALUE CLAUSE -- CLAUSE IGNORED
00014  77  ALPH-3           PIC  XXX      VALUE 123.                        DSFTST
00015  01  GROUP-AREA.                                                      DSFTST
00016      03  NUMBERUNO    PIC S999     VALUE 1.                          DSFTST
00017                                                                       DSFTST
00018  PROCEDURE DIVISION.                                                   DSFTST
00019  START-PROG.                                                           DSFTSTC
?E          CBL-0326 E   CONSOLE IS NOT A DEFINED NAME
00020      ACCEPT NEXT-KEY FROM CONSOLE.                                     DSFTST
?E          CBL-0331 E   UMBERNUO IS NOT DEFINED
00021      IF NEXT-KEY LESS THAN UMBERNUO  GO TO END-OF-PROGRAM.             DSFTST
?E          CBL-0443 E   GROUP-AREA INVALID HERE -- NOT NUMERIC
00022      ADD 1 TO GROUP-AREA.                                              DSFTST
00023      MOVE NUMBERUNO TO NOM-KEY.                                        DSFTST
00024      ADD 1 TO NUMBERUNO.                                               DSFTST
00025      GO TO START-PROG.                                                 DSFTST
00026  END-OF-PROGRAM.                                                       DSFTST
00027      STOP RUN.                                                         DSFTST
??SYS-3130 I DSFTST MODULE'S MAIN STORAGE SIZE IS
??           2924 DECIMAL
??SYS-3131 I 0000 IS THE START CONTROL ADDRESS OF THIS MODULE
??SYS-3134 I DSFTST MODULE IS CATALOGED AS A LOAD MEMBER
??           AGLIB    IS THE LIBRARY NAME
??               13 TOTAL NUMBER OF LIBRARY SECTORS
```

Figure 9-21. Diagnosed Source File Listing

Chapter 10. Ideographic Support

The following information describes ideographic support in COBOL. In order for COBOL to successfully process ideographic data, you must also have the ideographic version of the SSP and ideographic-capable input and output devices.

An ideographic character is a pictogram or graphic that requires 2 bytes of storage—in contrast with an alphanumeric character that requires only 1 byte of storage.

Ideographic support allows the COBOL compiler to process IBM-supplied or user-defined ideographic character sets. The basic ideographic character set contains 3707 characters consisting of 3226 Kanji characters and 481 additional characters. The basic ideographic character set is defined in hardware.

Basically, the fact that data is ideographic is transparent to System/34 COBOL. You must ensure that the ideographic data is processed properly by your program.

How to Specify that You Have Ideographic Literals

To indicate that you have ideographic literals, specify the GRAPHIC keyword on the PROCESS statement:

Entry	Explanation
GRAPHIC	GRAPHIC indicates to the compiler that ideographic literals can be present in the program.
NOGRAPH	NOGRAPH is the default. NOGRAPH indicates that no ideographic literals are present in the program.

The PROCESS statement is described in Chapter 9.

The Rules for Ideographic Literals

Ideographic literals follow the rules for nonnumeric literals, with the following additional requirements:

- A shift-out (S/O) control character (hex 0E) must immediately follow the apostrophe. The S/O control character indicates the start of a string of ideographic characters.
- The ideographic literal must end with a shift-in (S/I) control character (hex 0F) followed immediately by an apostrophe. The S/I control character indicates the end of a string of ideographic characters.

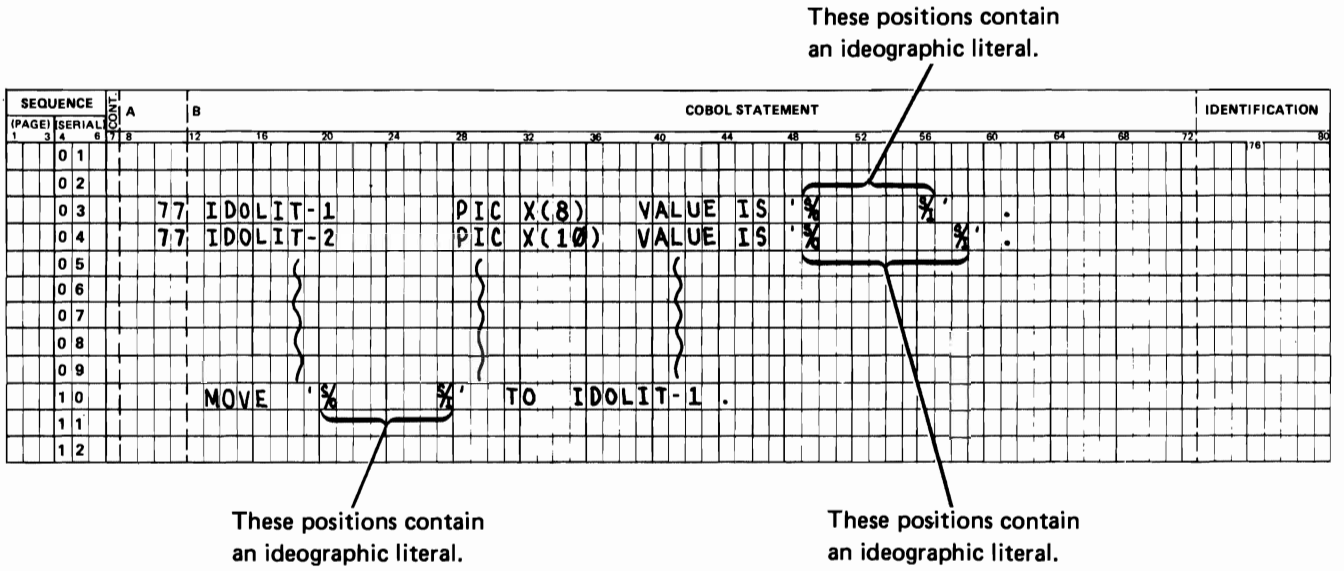
Note: In this manual, it is assumed that you are using apostrophes as the delimiters for literals. For a description of how to specify that you want to use double quotation marks as the delimiters for literals, see the description of the QUOTE/APOST option of the PROCESS statement in Chapter 9.

Any ideographic character can be entered in an ideographic literal. Each ideographic character has a 2-byte hex representation. (An ideographic blank also occupies 2 bytes.) An ideographic literal should consist of only ideographic characters; mixing ideographic characters and standard alphanumeric characters in the same literal causes the literal to be checked as an alphanumeric literal.

Because each ideographic character occupies 2 storage positions, any field that can contain ideographic data should be defined to have an even number of positions. Because an ideographic literal must contain both the S/O and S/I control characters, the minimum length of an ideographic literal is 2 positions. The maximum length of an ideographic literal is 120 positions, which includes the S/O and S/I control characters. Therefore, a maximum of 59 ideographic characters can be coded in a COBOL ideographic literal.

Examples of Ideographic Literals

The following are examples of ideographic literals:



Compiler Checking of Ideographic Literals

When the keyword GRAPHIC is specified on the PROCESS statement and the compiler finds a literal that begins with an apostrophe immediately followed by the S/O control character, the compiler checks for a valid ideographic literal. The following conditions cause a literal to be diagnosed as an invalid ideographic literal:

- An odd number of bytes are found between the S/O and S/I control characters.
- The terminating S/I control character is not immediately followed by an apostrophe.
- The ideographic literal spans multiple source lines and does not follow the rules for continuation of ideographic literals (for more information on continuation of ideographic literals, see *How to Specify Continuation of Ideographic Literals,* below).

If a literal is found to be an invalid ideographic literal, it is processed by the compiler as a nonnumeric literal.

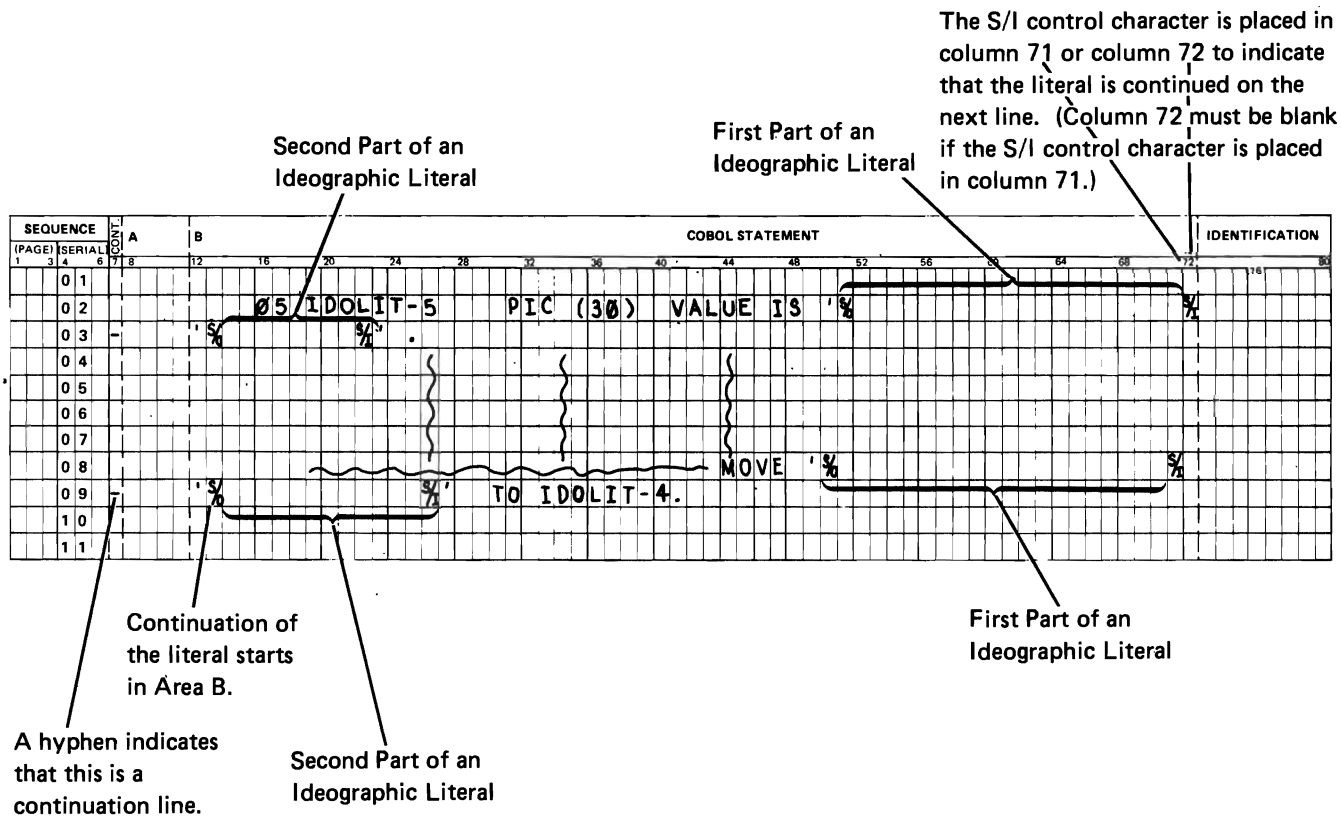
How to Specify Continuation of Ideographic Literals

When there is not room for an ideographic literal on one line of a COBOL coding form, do all of the following to indicate that the literal is continued:

- Place a S/I control character in column 71 or column 72 of the continued line.
- Place a - (hyphen) in column 7 (the continuation area) of the next line.
- Start the continuation line(s) with an apostrophe.
- Continue the literal with a S/O control character and the rest of the literal in Area B of the continuation line(s).

The S/I and S/O control characters that are used to indicate continuation are not counted in the length of the ideographic literal; the initial and final shift characters *are* counted.

The following are examples of continued ideographic literals:



Testing for Ideographic Support

To test whether ideographic support is available to your program, you can reference the ATTRIBUTE-DATA mnemonic-name with an ACCEPT statement. A description of the ATTRIBUTE-DATA mnemonic-name and an example of using the ACCEPT statement are in Chapter 7 under *SPECIAL-NAMES Paragraph*.

The following values are returned in the attribute record for a display station:

Byte 9 Display type
- A Alphanumeric or Katakana
- I Ideographic

Byte 10 Keyboard type
- A Alphanumeric or Katakana
- I Ideographic

Byte 11 Sign-on mode
- A Alphanumeric or Katakana
- I Ideographic

Bytes 12-16 Reserved

SUBROUTINES THAT HANDLE IDEOGRAPHIC DATA

COBOL provides two subroutines to handle ideographic data. These subroutines are necessary to move ideographic data to a field of a different length and insert or remove the S/O and S/I control characters.

System/34 requires that ideographic data be enclosed in the S/O and S/I control characters. Because you might need to send ideographic data to or receive ideographic data from another system that does not require the S/O and S/I control characters, COBOL provides two subroutines:

- CBINST–This subroutine moves ideographic data and inserts the S/O and S/I control characters.
- CBREMV–This subroutine moves ideographic data and removes the S/O and S/I control characters.

Both subroutines have the same calling sequence:

```
CALL {'CBINST' } USING data-name-1, data-name-2,
     {'CBREMV'}
       data-name-3, data-name-4, data-name-5
```

The data-names are elementary items with the following definition:

data-name	Definition	Purpose
data-name-1	PIC X( ) ...	Sending field
data-name-2	PIC X( ) ...	Receiving field
data-name-3	PIC X(1) ...	Return code
data-name-4	PIC 999 USAGE IS COMP	Length of sending field
data-name-5	PIC 999 USAGE IS COMP	Length of receiving field

The maximum length of the sending and receiving fields is 256 bytes.

Move Ideographic Data and Insert Control Characters–CBINST

CBINST is a move and edit subroutine that moves the contents of one field into another field. If the S/O and S/I control characters are not found in the first and last positions of the field to be moved, CBINST inserts them into the field when it is moved.

If you want the receiving field to contain all the data that is in the sending field, you must specify a receiving field length that is 2 positions longer than the length of the sending field. The 2 extra positions are to hold the S/O and S/I control characters. If you specify a receiving field that is longer than the sending field plus 2, the data is padded on the right when it is moved. If the receiving field is specified either longer or shorter than the sending field plus 2 positions, the S/I control character is still placed in the correct position (the rightmost position).

Subroutine CBINST produces return codes to indicate the status of the move. The following lists these return codes and their meanings:

Return Code	Explanation
0	Move executed; no errors.
1	Move executed; padding occurred to the left of the S/I control character.
2	Move executed; data truncated to the left of the S/I control character.
3	Move executed; S/O and S/I control characters already present.
4	Move not executed. Either odd field length found, length of zero found, length greater than 256, or invalid character found in field length.

If more than one return code can be issued, only the highest return code is returned.

Move Ideographic Data and Remove Control Characters–CBREMV

CBREMV is a move and edit subroutine that moves the contents of one field to another field. If the S/O and S/I control characters are found as the first and last characters in the field, CBREMV removes them.

If you want the receiving field to contain all the data that was present in the sending field, you must specify a receiving field length that is 2 positions less than the length of the sending field. This allows 2 positions for each ideographic character, while removing the S/O and S/I control characters (and the 2 positions they occupied). If you specify a receiving field longer than the sending field minus 2 positions, all the data from the sending field is moved and the receiving field is padded on the right with blanks. If the receiving field is shorter than the sending field minus 2 positions, the data being moved is truncated on the right.

Subroutine CBREMV produces return codes to indicate the status of the move. The following lists these return codes and their meanings:

Return Code	Explanation
0	Move executed; no errors.
1	Move executed; padding occurred.
2	Move executed; data truncated.
3	Move executed; S/O and S/I control characters not found in sending field.
4	Move not executed. Either odd field length found, length of zero found, length greater than 256, or invalid character found in field length.

If more than one return code can be issued, only the highest return code is returned.

Appendix A. Compiler Messages

Compiler-generated messages indicate conditions encountered during program compilation. The messages describe an invalid use of COBOL syntax or a violation of system requirements. Object-time messages and displayed compiler messages appear in the *Displayed Message Guide*.

Error messages are listed here in order by message number and are described using the following format:

Identifying Code	**Message**
CBL–0017 C	NAME/name/BEGINS OR ENDS WITH HYPHEN--ACCEPTED
What the error is and what caused it	*Explanation*: A COBOL name may have embedded hyphens, but must not begin or end with a hyphen.
What the compiler did as a result	*Compiler Action:* The name and all references to it are accepted.
What the programmer can do to correct it	*Programmer Response:* Probable user error. Replace the name and all references to the name with a name that does not begin or end with a hyphen. (Phase: TEXT)

Two general types of messages are supplied with source diagnostics produced at compile time.

1. In the first type, an assumption is made and/or a default is taken. By referencing the statement number supplied with the message, the programmer can read the explanation in this appendix, and the meaning of the message as it applies to the condition should be clear.
2. In the second type, no assumptions are made and/or no defaults are taken. Additional text describing the condition is given at the time the message is produced on the printer.

Variable content of diagnostic messages is always included within a pair of slash symbols (/ . . . /).

The compiler always attempts to provide full diagnostics of all source text in the program, even when errors have been discovered. If the compiler cannot continue on a given statement because no logic exists to place the remainder of the statement in proper context, the message states that the compiler cannot continue and that it will ignore the rest of the statement. If this occurs, the programmer should examine the entire statement.

DIAGNOSTIC LEVELS

System/34 COBOL provides the following three levels of diagnostics:

Code	**Meaning**
W	Warning. A possible error was detected in the source program. Examine the statement to ensure that no coding mistake was made.
C	Conditional. An error was detected, and the compiler is taking a recovery that may (conditionally) yield the correct code desired by the programmer.
E	Error. A serious error was made. Execution of the program should not be attempted.

Note: W and C messages are suppressed when the PROCESS statement FLAGE option is used (see *PROCESS Statement* in Chapter 9).

CBL–0002 C INVALID CHARACTER IN COLUMN 7--ASSUMED BLANK

Explanation: A line may contain only D, asterisk, slash, hyphen, or blank in column 7. One of the lines in this statement contains an invalid character in column 7.

Compiler Action: The line is processed as if column 7 contained a blank.

Programmer Response: Probable user error. Place a valid character in column 7 before recompiling. (Phase: COPY, TEXT)

CBL–0003 C CONTINUATION LINE INVALID AT THIS POINT--COLUMN 7 ASSUMED BLANK

Explanation: A continuation line (hyphen in column 7) must be immediately preceded by a normal line (blank in column 7) or by another continuation line.

Compiler Action: The line is processed as if column 7 contained a blank.

Programmer Response: Probable user error. Either place statements in proper sequence, include any omitted statements, or remove any interspersed comment lines before recompiling. (Phase: COPY, TEXT)

CBL–0004 W CONTINUATION LINE BLANK--LINE IGNORED

Explanation: A continuation line (hyphen in column 7) contains no source code in columns 8 through 72.

Compiler Action: The continuation line is ignored.

Programmer Response: Probable user error. Either continue preceding line on continuation line, remove continuation line, or replace the hyphen in column 7 with a blank. (Phase: COPY, TEXT)

CBL–0005 C FIRST CHARACTER ON CONTINUATION LINE IN AREA A--LINE ACCEPTED

Explanation: Any sentence or entry that requires more than one line must be continued by starting the subsequent line(s) in Area B.

Compiler Action: The continuation line is processed as if it started in Area B.

Programmer Response: Probable user error. Begin the continuation line(s) in Area B. (Phase: COPY, TEXT)

CBL–0006 C FIRST CHARACTER ON CONTINUATION LINE NOT QUOTE(S)--LITERAL TERMINATED

Explanation: An apostrophe placed anywhere in Area B must precede the continuation of a literal. An apostrophe must be represented by the character (''), if the QUOTE option was specified, or by the character (') if the APOST option was specified. If no option was specified, the character (') is required.

Compiler Action: The literal is terminated at the end of the line in which it appears. The continuation line is processed as if it contained a blank in column 7.

Programmer Response: Probable user error. Ensure that an apostrophe precedes continuation of literal in Area B before recompiling. (Phase: COPY, TEXT)

CBL–0007 C NO CONTINUATION OF NONNUMERIC LITERAL--LITERAL TERMINATED

Explanation: When a nonnumeric is continued from one line to another, a hyphen must be placed in column 7 of the continuation line.

Compiler Action: The literal is terminated at the end of the line in which it appears.

Programmer Response: Probable user error. End the literal on the previous line with an apostrophe, or continue the literal by placing a hyphen in column 7 of the continuation line (and an apostrophe before the continuation of the literal in Area B.) (Phase: COPY, TEXT)

CBL–0008 E SIZE OF NONNUMERIC LITERAL EXCEEDS 120--LITERAL TRUNCATED

Explanation: A nonnumeric literal must be composed of from 1 to 120 EBCDIC characters (excluding the quotation mark) enclosed in apostrophes.

Compiler Action: The nonnumeric literal is truncated to 120 characters.

Programmer Response: Probable user error. Ensure that the nonnumeric literal contains no more than 120 characters before recompiling. (Phase: COPY, TEXT)

CBL–0009 C EMPTY NONNUMERIC LITERAL--ONE BLANK ASSUMED

Explanation: At least one character, blank or nonblank, must appear between the opening apostrophe and the closing apostrophe of a literal. A literal cannot be specified by apostrophes in consecutive columns.

Compiler Action: The literal is assumed to contain one blank.

Programmer Response: Probable user error. Specify contents of nonnumeric literal before recompiling. (Phase: TEXT)

CBL–0010 C INVALID ITEM IN AREA A--ACCEPTED AS IF IN AREA B

Explanation: Area A is reserved for the beginning of division headers, section-names, paragraph-names, the FD, and level numbers.

Compiler Action: The item is assumed to begin in Area B.

Programmer Response: Probable user error. Either begin indicated item in Area B, correct item if it was intended to be a valid Area B item, or place the item in the correct division. (Phase: COPY, TEXT)

CBL–0011 E INVALID NUMERIC LITERAL/insert/--IGNORED

Explanation: The given item is not a valid numeric literal although it contains numeric characters.

Compiler Action: The indicated item is discarded.

Programmer Response: Probable user error. Ensure that the indicated item is a valid word, literal, or other language element. If it was intended to be a literal, check for a misplaced or invalid sign, a misplaced decimal point, or an invalid decimal-point character. (The period is valid unless DECIMAL POINT IS COMMA was specified, in which case the comma is valid.) (Phase: TEXT)

CBL–0012 E COPY INVALID WITHIN A COPY--IGNORED

Explanation: A COPY statement may not appear within source text that is itself being included in a COPY statement.

Compiler Action: The nested COPY clause is ignored.

Programmer Response: Probable user error. Correct the library member before recompiling. (Phase: COPY)

CBL–0013 W 'nnnnn' SEQUENCE ERROR(S) IN PROGRAM

Explanation: The numbers in the source statement sequence fields (columns 1 through 6) must be in ascending sequence (except for a blank field and for statements included via the COPY statement). The number of occurrences of nonascending sequence is included in the diagnostic.

Compiler Action: Each statement out of sequence is indicated by an S on the source listing. Otherwise, all compiler processing is the same.

Programmer Response: Probable user error. Rearrange the source programs so that the statements are in sequence. If they are in proper sequence, renumber the source statements in the sequence number field (columns 1 through 6). (Phase: COPY)

CBL–0014 E INVALID CURRENCY SIGN LITERAL--IGNORED

Explanation: The currency sign literal may be any EBCDIC character except the following:

*	Multiplication sign
+	Plus sign
-	Minus sign or hypen
=	Equal sign
/	Slash or division sign
.	Period
;	Semicolon
(	Left parenthesis
)	Right parenthesis
,	Comma
' or "	Apostrophe or quotation mark
Space	Space or blank
Numeric	0-9
Alphabetic	A, B, C, D, L, P, R, S, V, X, Z

Compiler Action: The character is not used by the compiler in scanning PICTURE clauses. The default character $ is used instead.

Programmer Response: Probable user error. Substitute a valid character in the CURRENCY clause before recompiling. (Phase: TEXT, DTA2)

CBL–0015 E INVALID PROGRAM-NAME IN CALL STATEMENT--IGNORED

Explanation: The program-name nonnumeric literal in a CALL statement must conform to the rules for forming a procedure-name.

Compiler Action: The literal is discarded.

Programmer Response: Probable user error. Correct the nonnumeric literal before recompiling. (Phase: TEXT)

CBL–0016 W INVALID IDEOGRAPHIC LITERAL DETECTED, NONNUMERIC LITERAL ASSUMED

Explanation: A literal was found that began with an apostrophe followed immediately by the S/O control character, but:

- An odd number of 1-byte characters were found between the S/O and the terminating S/I control characters.
- The terminating S/I control character was not immediately followed by an apostrophe.
- The ideographic literal spans multiple source lines and does not follow the rules for continuation of ideographic literals.

Compiler Action: The literal is processed as if it were a nonnumeric literal.

Programmer Response: Probable user error. Correct the literal so that it conforms to the rules for an ideographic literal. (Phase: TEXT)

CBL–0017 C NAME/name/BEGINS OR ENDS WITH HYPHEN--ACCEPTED

Explanation: A COBOL name may have embedded hyphens, but it must not begin or end with a hyphen.

Compiler Action: The name and all references to it are accepted.

Programmer Response: Probable user error. Replace the name and all references to the name with a name that does not begin or end with a hyphen. (Phase: TEXT)

CBL–0018 C /literal/CHARACTER MAY NOT PRECEDE /literal/ CHARACTER--ACCEPTED

Explanation: In COBOL, certain characters may not precede other characters; for example, a plus sign may not precede an alphabetic character.

Compiler Action: If either of the characters is a space, the compiler assumes the space to be omitted. If neither is a space, the compiler attempts to construct a valid item or items. If no other diagnostics are listed for the statement, the recovery was probably successful.

Note: The following are possible characters and their meaning:

.	Period
(	Left parenthesis
)	Right parenthesis
,	Comma or semicolon
*	Multiplication (*) or exponentiation (**) characters
/	Slash (division sign)
+	Plus sign
-	Minus sign or hyphen
=/GREATER/ LESS THAN	Equal sign, greater than, or less than
Space(s)	Spaces or blanks
' or "	Apostrophe or quotation mark
Numeric	0, 1, 2, . . . 99
Alphabetic	A, B, C, . . . Z, or a character not in the COBOL character s

Programmer Response: Probable user error. Correct the order of the indicated characters before recompiling. (Phase: TEXT)

CBL–0019 C /insert/IS INVALID GROUP OF CHARACTERS--IGNORED

Explanation: The indicated continuous string of characters does not conform to the rules for forming a COBOL language element.

Compiler Action: The indicated character string is not processed by the compiler.

Programmer Response: Probable user error. Correct the character string so it is a valid COBOL item before recompiling. (Phase: TEXT)

CBL–0020 E /name or item/EXCEEDS 30 CHARACTERS--TRUNCATED

Explanation: A character string that represents a PICTURE or a name must not be longer than 30 characters.

Compiler Action: The character string is processed as though it contained only its first 30 characters.

Programmer Response: Probable user error. If the item is a PICTURE, use the repetition factor (nnn) to reduce the length of the character string. If the item is a name, shorten it and all references to it to a unique name of 30 or fewer characters. (Phase: TEXT)

CBL–0021 E COPY STATEMENT NOT TERMINATED WITH PERIOD AND SPACE--COPY IGNORED

Explanation: A COPY statement must be terminated by a period.

Compiler Action: The COPY statement is ignored.

Programmer Response: Probable user error. Enter a period and spaces at the end of the COPY statement before recompiling. (Phase: COPY)

CBL–0023 E LIBRARY MEMBER NOT FOUND--COPY IGNORED

Explanation: The first eight characters of the library-name in a COPY statement (after automatic compiler conversion, when necessary) must correspond to an entry in the user's source library.

Compiler Action: The COPY statement is ignored.

Programmer Response: Probable user error. Either correct misspelled library-name, ensure that member is in the source library, insert a LIBRARY option if the source library is not in the same library as the COBOL compiler, or use the IN/OF library-name option of the COPY statement. (Phase: COPY)

CBL–0024 E INVALID LIBRARY-NAME FORMAT--COPY IGNORED

Explanation: The library-name in a COPY statement must conform to the rules for forming a COBOL procedure-name.

Compiler Action: The COPY statement is ignored.

Programmer Response: Probable user error. Correct the spelling of the library-name before recompiling. (Phase: COPY)

CBL–0025 W INVALID ELEMENT/element/

Explanation: In the PROCESS statement, a character or set of characters is unexpected in relation to its preceding items.

Compiler Action: If the element is the first or only word of another option, the previous option is discontinued and option processing begins with the element. If the element does not start an option, the element is discarded, and option processing for the previous option continues.

Programmer Response: Probable user error. Correct the syntax or spelling of the option before recompiling. (Phase: COPY)

CBL–0026 W INVALID OPTION AT/keyword/--OPTIONS TERMINATED

Explanation: In the PROCESS statement, a COBOL keyword that is not an option has been encountered, but an option was still being processed when this occurred. This error occurs if an incomplete option precedes a COBOL source statement.

Compiler Action: The option that precedes the indicated keyword is discontinued. All options that are processing are also terminated. The compiler begins its scan for normal COBOL source statements with the indicated keyword.

Programmer Response: Probable user error. Correct the option that immediately precedes the indicated keyword before recompiling. (Phase: COPY)

CBL–0027 C IDENTIFICATION DIVISION MISSING

Explanation: The first COBOL source statement must be IDENTIFICATION DIVISION.

Compiler Action: None.

Programmer Response: Probable user error. Insert the Identification Division header before the COBOL source statements (but after the PROCESS statement, if any) or correct the spelling or location of the word IDENTIFICATION. (Phase: TEXT)

CBL–0028 C PARAGRAPH OUT OF SEQUENCE--ACCEPTED

Explanation: Paragraphs in the Identification Division must be in this order: PROGRAM ID, AUTHOR, INSTALLATION, DATE-WRITTEN, SECURITY.

Compiler Action: None.

Programmer Response: Probable user error. Place the indicated paragraph in the proper sequence. (Phase: TEXT)

CBL–0029 C DUPLICATE ELEMENT--ACCEPTED

Explanation: A paragraph-name for the Identification Division and the Identification Division header may appear only once.

Compiler Action: The current item overrides the previous one.

Programmer Response: Probable user error. Delete the duplicate element. (Phase: TEXT)

CBL–0030 C INVALID ITEM/item/--RESTARTING AT NEXT KEYWORD IN AREA A

Explanation: The listed item is unexpected in the Identification Division.

Compiler Action: If the item is a keyword in Area A, the scan resumes with the item; if not, the scan resumes at the next keyword in Area A.

Programmer Response: Probable user error. Correct the syntax of the paragraph or division header in error, or move the indicated item to its proper location in the source program. (Phase: TEXT)

CBL–0031 C PROGRAM-ID MISSING

Explanation: The PROGRAM-ID paragraph is required within the Identification Division.

Compiler Action: The program is processed as though the program-name CBLOBJ had been specified.

Programmer Response: Probable user error. Place a PROGRAM-ID paragraph after the Identification Division header. (Phase: TEXT)

CBL–0032 C PERIOD AND SPACE NOT FOUND--ACCEPTED

Explanation: COBOL paragraph and division headers of the Identification Division must end with a period followed by at least one space.

Compiler Action: Processing continues as though a period and space had been specified.

Programmer Response: Probable user error. Insert a period and space after the header. (Phase: TEXT)

CBL–0033 C ELEMENT NOT IN AREA A--ACCEPTED

Explanation: Division and paragraph headers of the Identification Division must begin in Area A (columns 1 through 7).

Compiler Action: The header is assumed to begin in Area A.

Programmer Response: Probable user error. Ensure that the header begins in Area A before recompiling. (Phase: TEXT)

CBL–0034 C INVALID PROGRAM-NAME--IGNORED

Explanation: The program-name must conform to the rules for forming a procedure-name.

Compiler Action: The name is ignored, and processing continues as though the program-name CBLOBJ had been specified.

Programmer Response: Probable user error. Correct the program-name before recompiling. (Phase: TEXT)

CBL–0035 C PARAGRAPH NOT IN IDENTIFICATION DIVISION--ACCEPTED

Explanation: The Identification Division header must precede all paragraphs for the Identification Division.

Compiler Action: The paragraph is accepted as though it were in the Identification Division.

Programmer Response: Probable user error. Place the Identification Division header before all source statements (and after the PROCESS statement, if any). (Phase: TEXT)

CBL–0036 C INVALID IDENTIFICATION DIVISION HEADER--ACCEPTED

Explanation: The word DIVISION is required after the word IDENTIFICATION in Area A.

Compiler Action: The statement is accepted as though it had been coded correctly.

Programmer Response: Probable user error. Correct the Identification Division header. (Phase: TEXT)

CBL–0105 C MISSING PERIOD--ACCEPTED

Explanation: All sentences in a COBOL program must be terminated with a period.

Compiler Action: The statement is accepted as if it were terminated by a period.

Programmer Response: Probable user error. Supply missing period before recompiling. (Phase: DTA2, DTA3)

CBL–0107 E SELECT CLAUSE FOR FILE INVALID OR MISSING--FILE IGNORED

Explanation: Either no SELECT clause was specified for the file or the SELECT clause had an invalid system-name.

Compiler Action: The file is ignored.

Programmer Response: Probable user error. Correct invalid SELECT clause before recompiling. (Phase: DTA3)

CBL–0108 C BLOCK CONTAINS CLAUSE IS ZERO--UNBLOCKED RECORDS ASSUMED

Explanation: The computed size of the BLOCK CONTAINS clause is zero.

Compiler Action: The records are assumed to be unblocked.

Programmer Response: Probable user error. Correct invalid BLOCK clause specification before recompiling. (Phase: DTA3)

CBL–0109 E BUFFER SIZE EXCEEDED DEVICE LIMIT--FILE IGNORED

Explanation: The largest record description for the file is larger than the maximum buffer size allowed.

Compiler Action: The file is ignored.

Programmer Response: Correct invalid record description before recompiling. (Phase: DTA2, DTA3)

CBL–0110 C MAXIMUM BLOCKSIZE EXCEEDED--UNBLOCKED ASSUMED

Explanation: A BLOCK CONTAINS clause has a block size greater than 9999.

Compiler Action: The compiler assumes that records are not blocked.

Programmer Response: Probable user error. Ensure that the block size is 9999 or less before recompiling. (Phase: DTA3)

CBL–0111 E MAXIMUM RECORD SIZE EXCEEDED--CLAUSE IGNORED

Explanation: The RECORD CONTAINS clause has a record size greater than 4096.

Compiler Action: The RECORD CONTAINS clause is ignored.

Programmer Response: Probable user error. Ensure that the record size is 4096 or less before recompiling. (Phase: DTA3)

CBL–0112 C INVALID ENVIRONMENT DIVISION HEADER--ACCEPTED

Explanation: The word DIVISION or a period and space are missing from the Environment Division header.

Compiler Action: The Environment Division header is assumed to be correct.

Programmer Response: Probable user error. Correct syntax of indicated statement before recompiling. (Phase: DTA2)

CBL–0113 E ENVIRONMENT DIVISION IN INVALID LOCATION

Explanation: The Environment Division must immediately follow the Identification Division.

Compiler Action: Processing continues at the next section or division. All statements up to the next section or division are ignored; at that point, processing continues.

Programmer Response: Probable user error. Supply valid Environment Division header before recompiling. (Phase: DTA2)

CBL–0114 E MORE THAN 25 FILES SPECIFIED--IGNORED

Explanation: The maximum of 25 files was exceeded.

Compiler Action: The file is ignored.

Programmer Response: Probable user error. Reduce the number of files in the program to 25 before recompiling. (Phase: DTA2)

CBL–0115 C ENVIRONMENT DIVISION MISSING--ACCEPTED

Explanation: No Environment Division header was found.

Compiler Action: The compiler assumes that there is no Environment Division.

Programmer Response: Include Environment Division header before recompiling. (Phase: DTA2, DTA3)

CBL–0117 E /keyword/NOT IN ENVIRONMENT DIVISION

Explanation: The indicated word does not appear in the Environment Division.

Compiler Action: The statement is ignored, and the compiler skips to the next section, paragraph, or division.

Programmer Response: Probable user error. Ensure that the indicated word is in the Environment Division before recompiling. (Phase: DTA2)

CBL–0118 C /keyword/PARAGRAPH OR SECTION IN INVALID LOCATION--ACCEPTED

Explanation: Paragraph-name or section header is not in proper sequence or is duplicated in the Environment Division.

Compiler Action: The paragraph-name or section header is accepted.

Programmer Response: Probable user error. Ensure that all paragraph-names and section headers are in proper sequence before recompiling. (Phase: DTA2)

CBL–0119 C INVALID CONFIGURATION SECTION HEADER--ACCEPTED

Explanation: The word SECTION or a period and space are missing from the Configuration Section.

Compiler Action: The Configuration Section header is accepted as if it were correct.

Programmer Response: Probable user error. Correct section header before recompiling. (Phase: DTA2)

CBL–0120 E /name or keyword/NOT IN CONFIGURATION SECTION

Explanation: The indicated name or keyword is not within the Configuration Section.

Compiler Action: The statement is ignored, and the compiler skips to the next paragraph section or division.

Programmer Response: Probable user error. Ensure that the indicated item is in the Configuration Section before recompiling. (Phase: DTA2)

CBL–0121 W INVALID OBJECT-COMPUTER OR SOURCE-COMPUTER NAME--IGNORED

Explanation: The name following SOURCE-COMPUTER or OBJECT-COMPUTER is not IBM-S34.

Compiler Action: The paragraph is treated as comments.

Programmer Response: Probable user error. Supply valid name before recompiling. (Phase: DTA2)

CBL–0122 C INVALID OBJECT-COMPUTER MEMORY SIZE CLAUSE--IGNORED

Explanation: The OBJECT-COMPUTER MEMORY SIZE clause is not an integer from 1 through 65536.

Compiler Action: The MEMORY SIZE clause is ignored (the size of the current compiler region is assumed).

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: DTA2)

CBL–0123 E INVALID FUNCTION-NAME IN SPECIAL-NAMES PARAGRAPH--IGNORED

Explanation: The function-name is not one of the following: CONSOLE, REQUESTOR, C01, CSP, UPSI-0, UPSI-1, UPSI-2, UPSI-3, UPSI-4, UPSI-5, UPSI-6, UPSI-7; or the specified clause was not a CURRENCY SIGN, DECIMAL-POINT, or an alphabetic-name clause.

Compiler Action: The name is ignored and the compiler skips to the next function-name or clause, if any.

Programmer Response: Probable user error. Correct the function-name or clause before recompiling. (Phase: DTA2)

CBL–0124 W INVALID SYNTAX IN SPECIAL-NAMES

Explanation: The word IS is missing from the ON or OFF STATUS clause in a SPECIAL-NAMES statement for a switch.

Compiler Action: The statement is ignored; the compiler scans for the next function-name in the SPECIAL-NAMES paragraph.

Programmer Response: Probable user error. Correct the invalid clause before recompiling. (Phase: DTA2)

CBL–0125 E INVALID MNEMONIC-NAME OR CONDITION-NAME IN SPECIAL-NAMES

Explanation: The mnemonic-name or condition-name in a SPECIAL-NAMES statement is not a valid data-name.

Compiler Action: The statement is ignored. The compiler scans for the next function-name in the SPECIAL-NAMES paragraph.

Programmer Response: Probable user error. Correct the clause before recompiling. (Phase: DTA2)

CBL–0126 C INVALID SYNTAX IN CURRENCY SIGN CLAUSE--ACCEPTED

Explanation: The word IS has been omitted from or is misplaced in the CURRENCY SIGN clause.

Compiler Action: The clause is assumed to be correct.

Programmer Response: Probable user error. Correct the syntax of the CURRENCY SIGN clause before recompiling. (Phase: DTA2)

CBL–0127 C DUPLICATE CLAUSE FOUND IN SPECIAL-NAMES--FIRST ONE ACCEPTED

Explanation: A duplicate clause is detected in the SPECIAL-NAMES statement. The first clause is used.

Compiler Action: The duplicate clause is ignored.

Programmer Response: Probable user error. Correct the syntax before recompiling. (Phase: DTA2)

CBL–0128 C INVALID DECIMAL-POINT IS COMMA CLAUSE--ACCEPTED

Explanation: Either the word IS or the word COMMA is missing from or misplaced in the DECIMAL-POINT IS COMMA clause.

Compiler Action: DECIMAL-POINT IS COMMA is assumed.

Programmer Response: Probable user error. Correct the syntax of the DECIMAL-POINT IS COMMA clause before recompiling. (Phase: DTA2)

CBL–0129 C INVALID INPUT-OUTPUT SECTION HEADER--ACCEPTED

Explanation: Either SECTION or period and space are missing from the Input-Output Section header.

Compiler Action: The header is assumed to be correct.

Programmer Response: Probable user error. Correct the invalid statement before recompiling. (Phase: DTA2)

CBL–0130 E /name or keyword/NOT IN INPUT-OUTPUT SECTION--IGNORED

Explanation: Either the I-O CONTROL paragraph or the FILE-CONTROL paragraph is not in the Input-Output Section.

Compiler Action: The paragraph is ignored. The compiler skips to the next section, paragraph, or division.

Programmer Response: Probable user error. Ensure that the paragraph is in the Input-Output Section before recompiling. (Phase: DTA2)

CBL–0131 C INVALID ITEM/name or keyword/IN FILE-CONTROL PARAGRAPH--ITEM IGNORED

Explanation: The indicated name or keyword is invalid as used in the FILE-CONTROL paragraph.

Compiler Action: The indicated item is ignored.

Programmer Response: Probable user error. Correct indicated item before recompiling. (Phase: DTA2)

CBL–0132 C INVALID ITEM/name or keyword/IN SELECT CLAUSE--ITEM IGNORED

Explanation: The indicated name or keyword is invalid as used in the SELECT clause.

Compiler Action: The indicated item is ignored.

Programmer Response: Probable user error. Correct the invalid name or keyword before recompiling. (Phase: DTA2)

CBL–0133 E INVALID FILE-NAME IN SELECT CLAUSE--CLAUSE IGNORED

Explanation: The word SELECT was not followed by a valid file-name. (A name is not a valid file-name if it is a reserved word, a duplicate of a data-name, or is not uniquely defined in a valid FD or SD.)

Compiler Action: The SELECT clause is ignored.

Programmer Response: Probable user error. Correct invalid file-name before recompiling. (Phase: DTA2)

CBL–0134 E INVALID SYSTEM-NAME IN ASSIGN CLAUSE--FILE IGNORED

Explanation: The format or combinations for system-name are incorrect.

Compiler Action: The file is ignored.

Programmer Response: Probable user error. Correct the system-name specification before recompiling. (Phase: DTA2)

CBL–0135 E MORE THAN ONE TRANSACTION FILE IN PROGRAM--FILE IGNORED

Explanation: A SELECT clause has already been processed for a TRANSACTION file. There may be only one TRANSACTION file in a program.

Compiler Action: The file is ignored.

Programmer Response: Probable use error. Delete this file from the program. (Phase: DTA2)

CBL–0136 E INVALID DATA ITEM IN CONTROL-AREA--CLAUSE IGNORED

Explanation: The data-name in the CONTROL-AREA clause was omitted or was invalid for one of the following reasons:

- The data-name was not unique.
- The data-name was less than 12 characters long.
- The data-name was not an alphanumeric item.
- The data-name was not in the Working-Storage or Linkage Section.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct the invalid data-name before recompiling. (Phase: DTA2)

CBL–0137 C INVALID ACCESS CLAUSE--SEQUENTIAL ASSUMED

Explanation: The word DYNAMIC, RANDOM, or SEQUENTIAL is missing from the ACCESS MODE clause.

Compiler Action: ACCESS IS SEQUENTIAL is assumed.

Programmer Response: Probable user error. Specify access mode before recompiling. (Phase: DTA2)

CBL–0138 E INVALID ORGANIZATION CLAUSE--SEQUENTIAL ASSUMED

Explanation: The word RELATIVE, SEQUENTIAL, INDEXED, or TRANSACTION is missing from or is misplaced in the ORGANIZATION clause.

Compiler Action: The compiler assumes ORGANIZATION IS SEQUENTIAL for any further processing of the affected file.

Programmer Response: Probable user error. Correct the syntax of the ORGANIZATION clause before recompiling. (Phase: DTA2)

CBL–0139 E INVALID DATA-NAME IN KEY CLAUSE--CLAUSE IGNORED

Explanation: The error was caused by one of the following: the data-name of the key is missing; the RELATIVE KEY data-name was not in Working-Storage, or is not a numeric DISPLAY or COMPUTATIONAL(COMP) data item; the RECORD KEY is not from 1 to 29 bytes long, or is an edited field; or the RECORD KEY is not within the record area of the file to which it belongs.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: DTA2)

CBL–0140 C INVALID RESERVE CLAUSE--ONE AREA ASSUMED

Explanation: An invalid value of zero or a non-integer was specified in the RESERVE clause.

Compiler Action: RESERVE 1 AREA is assumed.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: DTA2)

CBL–0141 E /name or keyword/CLAUSE DUPLICATED IN SELECT CLAUSE--FIRST ONE ACCEPTED

Explanation: The indicated name or keyword has been duplicated in another SELECT clause.

Compiler Action: The first occurrence is accepted, and all duplications are ignored.

Programmer Response: Probable user error. Delete the duplicated name or keyword before recompiling. (Phase: DTA2)

CBL–0142 E INVALID NUMBER OF FORMATS IN TRANSACTION FILE ASSIGN--32 ASSUMED

Explanation: The number of formats in the TRANSACTION file system-name must not be greater than 32.

Compiler Action: The compiler assumes 32 formats for any further processing of the affected file.

Programmer Respone: Probable user error. Correct the invalid system-name before recompiling. (Phase: DTA2)

CBL–0143 E NO RELATIVE KEY FOR RANDOM FILE--FILE PROCESSED WITHOUT KEY

Explanation: The RELATIVE KEY clause must be specified for random access direct disk files.

Compiler Action: Results at execution time are unpredictable.

Programmer Response: Probable user error. Specify RELATIVE KEY before recompiling. (Phase: DTA2)

CBL–0144 E INVALID KEY TYPE FOR ACCESS METHOD--KEY IGNORED

Explanation: A RELATIVE or RECORD KEY has been specified for the wrong disk file organization or for a nondisk file.

Compiler Action: The KEY clause is ignored.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: DTA2)

CBL–0145 E NO ASSIGN CLAUSE IN FILE-CONTROL ENTRY--FILE IGNORED

Explanation: All files used in a program (named in a SELECT clause) must be assigned to an external medium by using the ASSIGN clause.

Compiler Action: The file is ignored.

Programmer Response: Probable user error. Supply ASSIGN clause before recompiling. (Phase: DTA2)

CBL–0146 E INVALID ORGANIZATION FOR TRANSACTION PROCESSING--TRANSACTION ASSUMED

Explanation: The organization clause is missing or specified RELATIVE, SEQUENTIAL, OR INDEXED, but the implementor-name is that of a TRANSACTION file.

Compiler Action: The compiler assumes ORGANIZATION IS TRANSACTION for any further processing of the affected file.

Programmer Response: Probable user error. Correct the invalid ORGANIZATION clause before recompiling. (Phase: DTA2)

CBL–0147 C /element/INVALID IN I-O-CONTROL PARAGRAPH--IGNORED

Explanation: The indicated element may not appear in the I-O-CONTROL paragraph.

Compiler Action: The invalid item is ignored.

Programmer Response: Probable user error. Correct invalid item before recompiling. (Phase: DTA2)

CBL–0148 C NUMBER OF SAME AREA CLAUSES EXCEEDS 15--CLAUSE IGNORED

Explanation: No more than 15 SAME AREA clauses or 15 SAME SORT/SORT-MERGE AREA clauses may appear in a program.

Compiler Action: The first 15 SAME AREA clauses or 15 SAME SORT/SORT-MERGE AREA clauses are recognized; any more than 15 are ignored.

Programmer Response: Probable user error. Ensure that no more than 15 SAME AREA clauses appear before recompiling. (Phase: DTA2)

CBL–0149 C INVALID FILE-NAME IN SAME AREA CLAUSE--FILE-NAME IGNORED

Explanation: An invalid file-name was found in the SAME AREA clause.

Compiler Action: The file-name is ignored.

Programmer Response: Probable user error. Ensure that file-name is valid in the SAME AREA clause. (Phase: DTA2)

CBL–0150 C FILE-NAME IN MULTIPLE SAME AREA CLAUSES--IGNORED

Explanation: Any file-name may appear in only one SAME AREA clause.

Compiler Action: The duplicated entry is ignored.

Programmer Response: Probable user error. Ensure that file-name appears in only one SAME AREA clause before recompiling. (Phase: DTA2)

CBL–0151 E FILE-NAME IN MULTIPLE RERUN CLAUSES--IGNORED

Explanation: A given file-name cannot be specified in more than one RERUN clause.

Compiler Action: The first RERUN clause for the indicated file-name is used.

Programmer Response: Probable user error. Ensure that a given file-name appears in only one RERUN clause before recompiling. (Phase: DTA2)

CBL–0152 E INVALID FILE-NAME IN RERUN CLAUSE--CLAUSE IGNORED

Explanation: The file-name was omitted or was invalid in a RERUN clause.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: DTA2)

CBL–0153 E INVALID NUMBER OF RECORDS IN RERUN CLAUSE--CLAUSE IGNORED

Explanation: The number of records in the RERUN clause must not exceed 32,767.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Ensure that the number of records does not exceed 32,767 before recompiling. (Phase: DTA2)

CBL–0154 E INVALID SYSTEM-NAME IN RERUN CLAUSE--CLAUSE IGNORED

Explanation: The system-name is not DISK-nnnnnnnn, where nnnnnnnn is an external name composed of from 1 to 8 characters.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: DTA2)

CBL–0155 C SYNTAX ERROR IN RERUN CLAUSE--ACCEPTED

Explanation: One or more of the required keywords is missing or misplaced.

Compiler Action: The clause is processed as if the required keywords were present.

Programmer Response: Probable user error. Include all required words in the RERUN clause before recompiling. (Phase: DTA2)

CBL–0156 E INVALID RESERVE CLAUSE--CLAUSE IGNORED

Explanation: The RESERVE clause is not valid in a SELECT statement for a TRANSACTION file.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct the invalid specification before recompiling. (Phase: DTA2)

CBL–0157 E INVALID CONTROL-AREA CLAUSE--CLAUSE IGNORED

Explanation: The CONTROL-AREA clause is not valid in a SELECT statement for a disk or printer file. (The CONTROL-AREA clause is valid only for a TRANSACTION file.)

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct the invalid specification before recompiling. (Phase: DTA2)

CBL–0158 C MORE THAN TWO RESERVE AREAS SPECIFIED--TWO ASSUMED

Explanation: More than two reserve areas were specified.

Compiler Action: Two reserve areas are assumed.

Programmer Response: Probable user error. Correct the syntax of the RESERVE clause before recompiling. (Phase: DTA2)

CBL–0159 E INVALID OPTIONAL CLAUSE--CLAUSE IGNORED

Explanation: The OPTIONAL clause is not valid in a SELECT statement for a TRANSACTION file.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct the invalid specification before recompiling. (Phase: DTA2)

CBL–0160 E NO RECORD KEY SPECIFIED FOR INDEXED FILE--FILE PROCESSED WITHOUT KEY

Explanation: An indexed file must have a RECORD KEY specification.

Compiler Action: The results at execution time are unpredictable.

Programmer Response: Probable user error. Specify RECORD KEY before recompiling. (Phase: DTA2)

CBL–0161 W EMPTY PARAGRAPH IN ENVIRONMENT DIVISION

Explanation: No statements have been found in a paragraph of the Environment Division.

Compiler Action: The empty paragraph header is ignored.

Programmer Response: Probable user error. Delete the paragraph header before recompiling. (Phase: DTA2)

CBL–0162 E INVALID ACCESS MODE CLAUSE--SEQUENTIAL ASSUMED

Explanation: The ACCESS MODE clause specifies DYNAMIC or RANDOM processing; both of these modes are incompatible with TRANSACTION file processing.

Compiler Action: The compiler assumes sequential.

Programmer Response: Probable user error. Correct the invalid specification before recompiling. (Phase: DTA2)

CBL – 0163 C INVALID LOCATION OF RELATIVE KEY PHRASE--ACCEPTED

Explanation: The RELATIVE KEY IS phrase, when used, should immediately follow the ACCESS MODE IS clause.

Compiler Action: The RELATIVE KEY IS phrase is processed as if it were part of the ACCESS MODE clause.

Programmer Response: Probable user error. Ensure that the RELATIVE KEY IS phrase is placed within the ACCESS MODE IS clause before recompiling. (Phase: DTA2)

CBL–0164 E INVALID FILE STATUS CLAUSE--CLAUSE IGNORED

Explanation: The keyword STATUS is either missing from the clause or is misplaced. The keyword FILE has been found, but the keyword STATUS does not follow it.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct the invalid clause before recompiling. (Phase: DTA2)

CBL–0165 E INVALID DATA ITEM IN FILE STATUS--CLAUSE IGNORED

Explanation: The data-name in the FILE STATUS clause was invalid, either because it was omitted or because of one of the following:

- The data-name was not unique.
- The data-name was not two characters long.
- The data-name was not an alphanumeric item.
- The data-name was not in the Working-Storage Section or Linkage Section.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct the invalid data-name before recompiling. (Phase: DTA2)

CBL–0166 C INVALID ELEMENT/name or keyword/IN STATEMENT--IGNORED

Explanation: An invalid element has been found in a section header or paragraph in the Environment Division.

Compiler Action: The remainder of the header is ignored.

Programmer Response: Probable user error. Correct indicated element before recompiling. (Phase: DTA2, DTA3)

CBL–0167 E INVALID ALPHABET NAME IN PROGRAM COLLATING SEQUENCE CLAUSE--IGNORED

Explanation: The alphabet name in the PROGRAM COLLATING SEQUENCE clause was invalid because it was omitted or because it was not unique.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct the invalid alphabet name before recompiling. (Phase: DTA2)

CBL–0168 C DUPLICATE CLAUSE IN OBJECT-COMPUTER PARAGRAPH

Explanation: A duplicate MEMORY SIZE, PROGRAM COLLATING SEQUENCE or SEGMENT LIMIT clause was found in the indicated paragraph.

Compiler Action: The first occurrence is accepted, and all duplications are ignored.

Programmer Response: Probable user error. Delete the duplicated clause(s) before recompiling. (Phase: DTA2)

CBL–0169 C SYNTAX ERROR IN SEGMENT-LIMIT CLAUSE--ACCEPTED

Explanation: The required keyword IS is either missing or misplaced.

Compiler Action: The clause is processed as if the required word were present.

Programmer Response: Probable user error. Include the word IS in the SEGMENT-LIMIT clause before recompiling. (Phase: DTA2)

CBL–0170 E INVALID SEGMENT-LIMIT CLAUSE--NONE ASSUMED

Explanation: The segment number specified on the SEGMENT-LIMIT clause is either missing or invalid. It is invalid if it is nonnumeric or not between 01 and 49, inclusive.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct the invalid segment number before recompiling. (Phase: DTA2)

CBL–0171 E INVALID CLAUSE IN SD FILE CONTROL ENTRY--CLAUSE IGNORED

Explanation: The only clauses allowed in an SD entry are the SELECT and ASSIGN clauses; any other clauses are invalid.

Compiler Action The clause(s) is ignored.

Programmer Response: Probable user error. Remove the invalid clause(s) before recompiling. (Phase: DTA2)

CBL–0172 E INVALID CLAUSE ON SD FILE DESCRIPTION--CLAUSE IGNORED

Explanation: Only a RECORD CONTAINS or DATA RECORDS clause is allowed to appear on an SD file description.

Compiler Action The clause(s) is ignored.

Programmer Response: Probable user error. Remove the invalid clause(s) before recompiling. (Phase: DTA3)

CBL–0173 E ALPHABET-NAME PREVIOUSLY DEFINED--CLAUSE IGNORED

Explanation: The ALPHABET-NAME was used in a previous ALPHABET-NAME clause.

Compiler Action: The clause is ignored and the compiler skips to the next FUNCTION-NAME or clause, if any.

Programmer Response: Probable user error. Correct the ALPHABET-NAME before recompiling. (Phase: DTA2)

CBL–0174 E PROGRAM COLLATING SEQUENCE NOT DEFINED

Explantion: The ALPHABET-NAME specified in the PROGRAM COLLATING SEQUENCE clause was not defined with an ALPHABET-NAME clause.

Compiler Action: The SPECIAL-NAMES paragraph is accepted as is.

Programmer Response: Probable user error. Add the required ALPHABET-NAME clause before recompiling. (Phase: DTA2)

CBL–0175 E CHARACTER PREVIOUSLY USED IN THIS COLLATING SEQUENCE--CHARACTER IGNORED

Explanation: The character value specified in the literal has been used previously in the same collating sequence.

Compiler Action: The value is ignored and the compiler continues with the next value/field.

Programmer Response: Probable user error. Correct the literal before recompiling. (Phase: DTA2)

CBL–0176 E INVALID PHRASE IN ALPHABET-NAME CLAUSE--PHRASE IGNORED

Explanation: The ALPHABET-NAME clause contains a phrase other than: STANDARD-1, NATIVE, or the literal phrase.

Compiler Action: The phrase is ignored and the compiler skips to the next FUNCTION-NAME or clause, if any.

Programmer Response: Probable user error. Correct the ALPHABET-NAME clause before recompiling. (Phrase: DTA2).

CBL–0177 E INVALID LITERAL IN ALPHABET-NAME CLAUSE--LITERAL IGNORED

Explanation: The field is one of the following:

- A numeric literal not in the range 1 through 256.
- A non-numeric literal of more than 1 character after the THROUGH or ALSO phrase.

Compiler Action: The field is ignored and the compiler skips to the next field.

Programmer Response: Probable user error. Correct the literal phrase before recompiling. (Phrase: DTA2)

CBL-0178 C REQUIRED SOURCE-COMPUTER KEY WORD NOT FOUND

Explanation: The compiler expects to find the required SOURCE-COMPUTER paragraph in the ENVIRONMENT DIVISION. This paragraph was either not specified correctly or was misplaced.

Compiler Action: The compiler assumes that SOURCE-COMPUTER. IBM-S34. was specified.

Programmer Response: Probable user error. Specify the SOURCE-COMPUTER paragraph correctly before recompiling. (Phase: DTA2)

CBL-0179 C REQUIRED OBJECT-COMPUTER KEY WORD NOT FOUND

Explanation: The compiler expects to find the required OBJECT-COMPUTER paragraph in the ENVIRONMENT DIVISION. This paragraph was either not specified correctly or was misplaced.

Compiler Action: The compiler assumes that OBJECT-COMPUTER. IBM-S34. was specified.

Programmer Response: Probable user error. Specify the OBJECT-COMPUTER paragraph correctly before recompiling. (Phase: DTA2)

CBL-0180 E RELATIVE KEY DATA ITEM LENGTH GREATER THAN 7--7 ASSUMED

Explanation: System/34 data management does not allow a relative (direct) file to have a key length greater than 7 bytes.

Compiler Action: The compiler assumes that the length of the RELATIVE KEY data item is 7 bytes.

Programmer Response: Probable user error. Ensure that the length of the RELATIVE KEY data item is 7 or less. (Phase: DTA2)

CBL-0186 E INVALID ACCESS MODE FOR PRINTER OR SEQUENTIAL DISK FILE--SEQUENTIAL ASSUMED

Explanation: The ACCESS clause for a printer or a sequential disk file was other than SEQUENTIAL.

Compiler Action: The ACCESS clause is ignored and SEQUENTIAL is assumed.

Programmer Response: Probable user error. Correct the ACCESS clause before recompiling. (Phase: DTA2)

CBL-0189 C SYNTAX ERROR IN APPLY CLAUSE--ACCEPTED

Explanation: The required keyword, CORE-INDEX, is either missing or misplaced.

Compiler Action: The clause is processed as if the required word were present.

Programmer Response: Probable user error. Include CORE-INDEX in the APPLY clause before recompiling. (Phase: DTA2)

CBL-0190 E INVALID FILE-NAME IN APPLY CORE-INDEX CLAUSE--FILE-NAME IGNORED

Explanation: Either no file-name was found or the file was not an indexed file.

Compiler Action: The invalid file-name is ignored, and any valid file-names are accepted.

Programmer Response: Probable user error. Provide a valid file-name before recompiling. (Phase: DTA2)

CBL-0191 E INVALID DATA-NAME IN APPLY CORE-INDEX CLAUSE--CLAUSE IGNORED

Explanation: The data-name in the APPLY CORE-INDEX clause was invalid, either because it was omitted or because of one of the following:

- It was not in the Working-Storage Section.
- It was an alphanumeric edited item.
- It was less than 3 or greater than 9999 bytes in length.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct invalid data-name before recompiling. (Phase: DTA2)

CBL-0192 C MORE THAN ONE CORE-INDEX FOR FILE--FIRST ONE ACCEPTED

Explanation: Only one APPLY CORE-INDEX clause is allowed per file.

Compiler Action: The first APPLY CORE-INDEX clause for the file is used.

Programmer Response: Probable user error. Ensure that only one APPLY CORE-INDEX clause is specified for the file before recompiling. (Phase: DTA2)

CBL–0200 E INVALID LOCATION OF FILE SECTION

Explanation: The File Section must begin on the line following the Data Division header.

Compiler Action: The File Section header is ignored, and the compiler skips to the next section header, FD, or data item.

Programmer Response: Probable user error. Ensure that the File Section begins on the line following the Data Division header before recompiling. (Phase: DTA1)

CBL–0201 E FD OR SD NOT IN FILE SECTION

Explanation: The File Section header must precede all file-control entries (FD, SD).

Compiler Action: The FD/SD is ignored, and the compiler skips to the next section header, FD, SD, or data item.

Programmer Response: Probable user error. Ensure that FD/SD is located in the File Section before recompiling. (Phase: DTA1)

CBL–0202 E INVALID FILE-NAME FOLLOWING FD OR SD--FILE IGNORED

Explanation: Each file description entry must consist of a level indicator FD or SD, followed by a unique file-name and a series of independent clauses.

Compiler Action: The file description is ignored.

Programmer Response: Probable user error. Ensure that the level indicators are followed by a valid file-name before recompiling. (Phase: DTA1)

CBL–0203 E INVALID OR MISSING NAME AFTER LEVEL NUMBER

Explanation: A data-name or the word FILLER must be the first word following the level number in each data description entry.

Compiler Action: The level item is ignored.

Programmer Response: Probable user error. Ensure that a data-name or the word FILLER is the first word following the level number before recompiling. (Phase: DTA1)

CBL–0204 E INVALID OCCURS CLAUSE--CLAUSE IGNORED

Explanation: The literals in the OCCURS clause are missing or do not have a value of from 1 to 32,768, the second literal is not greater than the first, the DEPENDING ON clause is not followed by a date-name, or one of the required keywords is missing.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct the literal(s) before recompiling. (Phase: DTA1)

CBL–0205 E INVALID REDEFINES CLAUSE--CLAUSE IGNORED

Explanation: No data-name was found, the clause was preceded by another clause, or the redefining item describes a variable length table.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct invalid clause before recompiling. (Phase: DTA1)

CBL–0206 E INVALID PICTURE CLAUSE--CLAUSE IGNORED

Explanation: The PICTURE literal is invalid.

Compiler Action: The data item is ignored.

Programmer Response: Probable user error. Correct invalid clause before recompiling. (Phase: DTA1)

CBL–0207 E INVALID LITERAL IN VALUE CLAUSE--CLAUSE IGNORED

Explanation: The error was caused by one of the following: a numeric literal was specified for a nonnumeric data item; the number of significant digits in the numeric literal exceeds the number of significant digits in the data item; the value is not within an allowable range for this data item; or an invalid literal or no literal followed the ALL literal option.

Compiler Action: The VALUE clause is ignored.

Programmer Response: Probable user error. Correct invalid literal before recompiling. (Phase: DTA1, DLIT)

CBL–0208 E INVALID LEVEL NUMBER--ITEM IGNORED

Explanation: Either the level number is not 01 through 47, 66, 77, or 88 or the level number of a group item is invalid.

Compiler Action: The data item is ignored.

Programmer Response: Probable user error. Supply correct level number before recompiling. (Phase: DTA1)

CBL–0209 E INVALID OBJECT OF REDEFINES--CLAUSE IGNORED

Explanation: The object of the REDEFINES clause is an undefined data-name, contains a REDEFINES clause, is not at the same level, or is separated from the statement containing the REDEFINES by a data item that is not part of a REDEFINES clause.

Compiler Action: The REDEFINES clause is ignored.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: DTA1)

CBL–0210 E NUMERIC ITEM EXCEEDS 18--ITEM IGNORED

Explanation: The numeric item has more than 18 digits.

Compiler Action: The numeric item is ignored.

Programmer Response: Probable user error. Ensure that the numeric item does not exceed 18 digits before recompiling. (Phase: DTA1)

CBL–0211 E VALUE EXCEEDS SIZE OF ITEM--CLAUSE IGNORED

Explanation: The number of digits to the left of the decimal point in the numeric literal is greater than the number of digits to the left of the decimal point in the data item.

Compiler Action: The VALUE clause is ignored, and the data item is not initialized.

Programmer Response: Probable user error. Correct size of item before recompiling. (Phase: DLIT)

CBL–0212 E SIZE OF EDITED ITEM EXCEEDS 127--ITEM IGNORED

Explanation: The size of a numeric edited data item may not exceed 127 characters (bytes) of storage.

Compiler Action: The item is ignored.

Programmer Response: Probable user error. Correct size of item before recompiling. (Phase: DTA1)

CBL–0213 E VALUE IN OCCURS ITEM--VALUE CLAUSE IGNORED

Explanation: OCCURS and VALUE clauses may not both be specified for a data item.

Compiler Action: The VALUE clause is ignored.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: DTA1)

CBL–0214 E INVALID PARENT OF VARIABLE LENGTH TABLE--TABLE IGNORED

Explanation: An item containing an OCCURS . . . DEPENDING ON clause is subordinate to an item containing either another OCCURS clause or a REDEFINES clause.

Compiler Action: The subordinate OCCURS clause is ignored.

Programmer Response: Probable user error. Ensure that no parent of the variable length table contains an OCCURS clause or REDEFINES clause or that the subordinate table is of fixed length. (Phase: DTA1)

CBL–0215 E NUMBER OF NESTED OCCURS EXCEEDS LIMIT--FIRST THREE ACCEPTED

Explanation: Only three nested levels of the OCCURS clause are allowed.

Compiler Action: All levels of the OCCURS clause beyond the third are ignored. Execution time results are unpredictable.

Programmer Response: Probable user error. Ensure that only three nested levels of the OCCURS clause are present before recompiling. (Phase: DTA1)

CBL–0216 E INVALID INDEX-NAME IN OCCURS CLAUSE--OPTION IGNORED

Explanation: INDEXED BY is not followed by a data-name.

Compiler Action: The INDEXED BY option is ignored.

Programmer Response: Probable user error. Ensure that INDEXED BY option is followed by a data-name before recompiling. (Phase: DTA1)

CBL–0217 E 77 LEVEL IN FILE SECTION--ITEM IGNORED

Explanation: Level-77 items may appear only in the Working-Storage Section or the Linkage Section.

Compiler Action: The item is ignored.

Programmer Response: Change the level number from 77 to 01 before recompiling. (Phase: DTA1)

CBL–0218 E CONDITIONAL VARIABLE MISSING--CONDITION-NAME IGNORED

Explanation: A condition-name (88) had no immediate qualifier.

Compiler Action: The condition-name (88) is ignored.

Programmer Response: Probable user error. Insert an immediate qualifier for the condition-name before recompiling. (Phase: DTA1)

CBL–0219 E NO VALUE IN CONDITION-NAME--ITEM IGNORED

Explanation: The VALUE clause was omitted from the condition-name (88).

Compiler Action: The condition-name is ignored.

Programmer Response: Probable user error. Supply VALUE clause before recompiling. (Phase: DTA1)

CBL–0220 E INVALID LOCATION OF WORKING-STORAGE SECTION

Explanation: The Working-Storage Section header was either duplicated or preceded by the Linkage Section header.

Compiler Action: The section header is ignored.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: DTA1)

CBL–0221 E INVALID LOCATION OF LINKAGE SECTION

Explanation: The Linkage Section header was duplicated.

Compiler Action: The second Linkage Section header is ignored.

Programmer Response: Probable user error. Delete second Linkage Section header before recompiling. (Phase: DTA1)

CBL–0222 E REDEFINES OF ITEM WITH OCCURS CLAUSE--REDEFINES IGNORED

Explanation: The item being redefined cannot contain an OCCURS clause or be subordinate to an item that contains an OCCURS clause.

Compiler Action: The REDEFINES clause is ignored.

Programmer Response: Probable user error. Remove the OCCURS clause from the description of the object of the REDEFINES clause, or eliminate the use of the REDEFINES clause before recompiling. (Phase: DTA1)

CBL–0223 E VALUE IN REDEFINES ITEM--VALUE CLAUSE IGNORED

Explanation: The VALUE clause may not be specified in a data description entry that contains a REDEFINES clause or in an entry that is subordinate to an entry containing a REDEFINES clause.

Compiler Action: The VALUE clause is ignored.

Programmer Response: Delete the invalid VALUE clause before recompiling. (Phase: DTA1)

CBL-0224 E NON-UNIQUE INDEX-NAME--IGNORED

Explanation: A data-name in an INDEXED BY option may not be defined more than once.

Compiler Action: The duplicate index-name in the INDEXED BY option is ignored.

Programmer Response: Probable user error. Make the index-name unique before recompiling. (Phase: DTA1)

CBL-0225 E PRECEDING ITEM/name/REDEFINES A SMALLER ITEM

Explanation: The indicated data item (other than an 01 item) redefines a smaller item.

Compiler Action: Sufficient space is allocated for the larger item. (Both the redefined and redefining fields will have the same left address.)

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: DTA1)

CBL-0226 E ITEM SIZE EXCEEDS 32,768--IGNORED

Explanation: An item of over 32,768 bytes was found.

Compiler Action: The item is ignored.

Programmer Response: Probable user error. Correct item size before recompiling. (Phase: DTA1)

CBL-0227 C INVALID INDICATOR CLAUSE--IGNORED

Explanation: The INDICATOR clause was specified for a data-item that is not Boolean.

Compiler Action: The INDICATOR clause is ignored.

Programmer Response: Probable user error. Correct the invalid specification before recompiling. (Phase: DTA1)

CBL-0228 E NON-UNIQUE FILE-NAME--FILE IGNORED

Explanation: A file description entry contains a file-name that was previously defined.

Compiler Action: The file is ignored.

Programmer Response: Probable user error. Ensure that each file has a unique name before recompiling. (Phase: DTA1)

CBL-0229 C INVALID HIERARCHY--ACCEPTED

Explanation: A level item does not have a preceding level-01 item; a variable length table is not the last structure within a record description; a level-66 item is not the last item in a record description.

Compiler Action: The items are accepted.

Programmer Response: Probable user error. Correct the invalid hierarchy before recompiling. (Phase: DTA1, DLIT)

CBL-0230 E INVALID OBJECT OF OCCURS . . DEPENDING ON CLAUSE

Explanation: The data description of the object of an OCCURS . . . DEPENDING ON clause does not define an elementary unedited numeric item, or it is defined within the structure of the variable length table.

Compiler Action: Processing continues with the invalid item; however, incorrect code will be generated.

Programmer Response: Probable user error. Correct the data description of the item before recompiling. (Phase: DLIT)

CBL-0231 E INVALID CLAUSE IN DATA DIVISION--FOUND/element/

Explanation: The indicated element has been found in the Data Division where its use is invalid.

Compiler Action: The element and all elements up to the next section header, level indicator, or level number are ignored.

Programmer Response: Move invalid item to proper location or delete before recompiling. (Phase: DTA1)

CBL–0232 E INVALID CLAUSE IN FD ENTRY--FOUND/element/

Explanation: The element found was not one of the following: BLOCK CONTAINS, RECORD CONTAINS, LABEL RECORDS, LINAGE, CODE-SET, or DATA RECORDS clause; a period, or a space.

Compiler Action: The element is ignored, and the remainder of the FD is processed.

Programmer Response: Probable user error. Delete invalid element before recompiling. (Phase: DTA3)

CBL–0233 E VALUE DEFINED IN ITEM WITH GROUP VALUE--GROUP VALUE ASSUMED

Explanation: If the VALUE clause is specified in an entry at the group level, it may not be specified at subordinate levels within this group.

Compiler Action: The group value is processed, and the lower level is ignored.

Programmer Response: Probable user error. Before recompiling, ensure that a VALUE clause does not appear both on the group level and on a level subordinate to the group level. (Phase: DTA1)

CBL–0234 C INVALID INDICATOR VALUE--CLAUSE IGNORED

Explanation: The value for an INDICATOR clause was not an integer between 1 and 99.

Compiler Action: The INDICATOR clause is ignored.

Programmer Response: Probable user error. Correct the incorrect value before recompiling. (Phase: DTA1)

CBL–0235 E VALUE CLAUSE INVALID IN SECTION--IGNORED

Explanation: A VALUE clause was specified for a non-level 88 item in the File or Linkage Section.

Compiler Action: The VALUE clause is ignored.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: DTA1)

CBL–0236 E NO PICTURE IN PRECEDING ELEMENTARY ITEM

Explanation: A higher-level item was found following a lower-level item with no PICTURE clause.

Compiler Action: The compiler substitutes a dummy item for the invalid item. The results at execution time are unpredictable.

Programmer Response: Probable user error. Provide PICTURE clause before recompiling. (Phase: DTA1)

CBL–0237 E INVALID CHARACTER IN PICTURE--ITEM IGNORED

Explanation: The categories of data that can be described with a PICTURE clause are made up of certain allowable combinations of characters in the COBOL character set. A character that is not allowed has been found in the PICTURE.

Compiler Action: The data item is ignored.

Programmer Response: Probable user error. Correct invalid PICTURE before recompiling. (Phase: DTA1)

CBL–0238 E USAGE AND PICTURE NOT COMPATIBLE--USAGE IGNORED

Explanation: An alphabetic, alphanumeric, or numeric edited PICTURE clause has usage other than DISPLAY.

Compiler Action: The USAGE clause is ignored.

Programmer Response: Probable user error. Specify USAGE IS DISPLAY before recompiling. (Phase: DTA1)

CBL–0239 E GROUP CONTAINS PICTURE CLAUSE--GROUP PICTURE IGNORED

Explanation: A group item may not have a PICTURE clause.

Compiler Action: The PICTURE clause is ignored in the group data item.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: DTA1)

CBL–0240 E INVALID VALUE IN GROUP ITEM

Explanation: A numeric literal has been found as the value for a group item.

Compiler Action: The VALUE clause is dropped, and the group item is not initialized.

Programmer Response: Probable user error. Ensure that the value specified is a figurative constant or a nonnumeric literal before recompiling. (Phase: DLIT)

CBL–0241 E ALPHANUMERIC VALUE EXCEEDS ITEM SIZE--VALUE CLAUSE IGNORED

Explanation: The nonnumeric literal in the VALUE clause must not exceed the size of the alphanumeric item.

Compiler Action: The VALUE clause is ignored, and the data item is not initialized.

Programmer Response: Probable user error. Ensure that the literal in the VALUE clause is less than or equal to the item size before recompiling. (Phase: DLIT)

CBL–0242 E FILE NAME APPEARS ON MORE THAN ONE SAME AREA CLAUSE

Explanation: One or more of the file-names appearing on this SAME AREA clause has appeared on at least one previous SAME AREA clause.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct the invalid file-name(s) before recompiling. (Phase: DTA2)

CBL–0243 E FILE NAME APPEARS ON MORE THAN ONE SAME RECORD AREA CLAUSE

Explanation: One or more of the file-names appearing on this SAME RECORD AREA clause has appeared on at least one previous SAME RECORD AREA clause.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct the invalid file-name(s) before recompiling. (Phase: DTA2)

CBL–0244 E FILE NAME(S) APPEARING ON SAME AREA CLAUSES DO NOT ALL APPEAR ON SAME RECORD AREA CLAUSE

Explanation: If one file from a SAME AREA clause appears on a SAME RECORD AREA clause, all files named on the SAME AREA clause must appear on the SAME RECORD AREA clause.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct the invalid file-name(s) before recompiling. (Phase: DTA2)

CBL–0245 E DUPLICATED CLAUSE IN DATA DESCRIPTION--FIRST ONE ACCEPTED

Explanation: The same clause occurred more than once in the same data description.

Compiler Action: The first clause is processed, and all duplications are ignored.

Programmer Response: Probable user error. Delete duplications before recompiling. (Phase: DTA1)

CBL–0246 E USAGE IN ITEM NOT SAME AS GROUP USAGE--GROUP USAGE USED

Explanation: The usage of an elementary item may not contradict the usage of a group to which the elementary item belongs.

Compiler Action: The group usage overrides the elementary usage.

Programmer Response: Probable user error. Correct usage in elementary item before recompiling. (Phase: DTA1)

CBL–0247 E LEVEL ITEMS IN FILE SECTION--NO PRECEDING FD OR SD

Explanation: A file description entry must precede the first level-01 in the File Section.

Compiler Action: Space is allocated for the level-01 item, but there is no corresponding file for I/O verbs.

Programmer Response: Probable user error. Insert a valid file description (FD or SD) before the level-01 data item, or remove the level-01 item before recompiling. (Phase: DTA1)

CBL–0248 E JUSTIFIED OR USAGE CLAUSE INVALID IN GROUP ITEM WITH VALUE--CLAUSE IGNORED

Explanation: A VALUE clause cannot be specified for a group containing items with descriptions including JUSTIFIED or USAGE.

Compiler Action: The JUSTIFIED or USAGE clause is ignored.

Programmer Response: Probable user error. Delete JUSTIFIED, USAGE, or VALUE clause before recompiling. (Phase: DTA1)

CBL–0249 E VALUE IN PRECEDING GROUP ITEM EXCEEDS GROUP ITEM SIZE

Explanation: The literal specified as a VALUE in a group item is too large.

Compiler Action: The VALUE clause is dropped, and the group item is not initialized.

Programmer Response: Probable user error. Ensure that the length of the literal is less than or equal to the size of the group item before recompiling. (Phase: DLIT)

CBL–0250 C KEY CLAUSE ON BOOLEAN ITEM--CLAUSE IGNORED

Explanation: ASCENDING/DESCENDING KEY clause was specified for a Boolean data-item.

Compiler Action: The ASCENDING/DESCENDING KEY clause is ignored.

Programmer Response: Probable user error. Remove the invalid option before recompiling. (Phase: DTA1)

CBL–0252 C INVALID DATA DIVISION HEADER--ACCEPTED

Explanation: The word DIVISION or a period and a space were omitted or misplaced in the Data Division header.

Compiler Action: The Data Division header is assumed to be correct.

Programmer Response: Probable user error. Supply proper Data Division specification before recompiling. (Phase: DTA1)

CBL–0253 C COMPUTED RECORD SIZE NOT EQUAL TO DEFINED RECORD SIZE

Explanation: The RECORD CONTAINS clause in the FD or SD entry contains an integer larger than the largest 01 record definition or smaller than the smallest 01 record definition for the file.

Compiler Action: The clause is treated as comments.

Programmer Response: Probable user error. Correct clause before recompiling. (Phase: DTA3)

CBL–0254 C INVALID WORD/CHARACTER(S) /element/ IN CLAUSE--IGNORED

Explanation: An invalid clause was found in a level item.

Compiler Action: The compiler skips to the next keyword or level item.

Programmer Response: Probable user error. Delete invalid element before recompiling. (Phase: DTA1)

CBL–0255 C INVALID USAGE CLAUSE--DISPLAY ASSUMED

Explanation: The USAGE clause must contain one of the following keywords: DISPLAY, COMPUTATIONAL(COMP), COMPUTATIONAL-3(COMP-3), COMPUTATIONAL-4(COMP-4), or INDEX.

Compiler Action: The usage is assumed to be DISPLAY.

Programmer Response: Probable user error. Specify valid USAGE clause before recompiling. (Phase: DTA1)

CBL–0256 C INVALID USE OF BLANK WHEN ZERO CLAUSE--IGNORED

Explanation: The BLANK WHEN ZERO clause may be specified only at the elementary level for numeric edited or numeric items; it may not be specified for level-88 data items and may not be used in conjunction with * in the PICTURE clause.

Compiler Action: The BLANK WHEN ZERO clause is ignored.

Programmer Response: Probable user error. Delete invalid BLANK WHEN ZERO clause before recompiling. (Phase: DTA1)

CBL–0257 C INVALID USE OF JUSTIFIED CLAUSE--IGNORED

Explanation: The JUSTIFIED clause may only be specified at the elementary level for nonnumeric unedited items.

Compiler Action: The JUSTIFIED clause is ignored.

Programmer Response: Probable user error. Remove the invalid JUSTIFIED clause before recompiling. (Phase: DTA1)

CBL–0258 C INVALID CLAUSE IN 88 LEVEL ITEM--IGNORED

Explanation: A clause other than VALUE was specified for a level-88 data item.

Compiler Action: The invalid clause is ignored.

Programmer Response: Probable user error. Delete invalid clause from level-88 item before recompiling. (Phase: DTA1)

CBL–0259 C INVALID CLAUSE IN INDEX DATA ITEM--IGNORED

Explanation: JUSTIFIED, BLANK, PICTURE, and VALUE clauses may not be specified for a data item with USAGE INDEX.

Compiler Action: The invalid clause is ignored.

Programmer Response: Probable user error. Delete invalid clause before recompiling. (Phase: DTA1)

CBL–0260 C REDEFINES OF ITEM WITH REDEFINES CLAUSE--REDEFINES ACCEPTED

Explanation: The object of the REDEFINES clause contains a REDEFINES clause.

Compiler Action: Both REDEFINES clauses are accepted.

Programmer Response: Probable user error. Ensure that the object of both REDEFINES clause is the same data name before recompiling. (Phase: DTA1)

CBL–0261 C OCCURS IN LEVEL 01 OR 77 ITEM--ACCEPTED

Explanation: The OCCURS clause may not be specified in a data description entry that is a level-01 or level-77 number.

Compiler Action: The OCCURS clause is accepted.

Programmer Response: Probable user error. Correct level number before recompiling. (Phase: DTA1)

CBL–0262 C REDEFINES IN 01 LEVEL IN FILE SECTION--IGNORED

Explanation: The REDEFINES clause should not be used in level-01 entries in the File Section because multiple 01's are an implicit redefinition.

Compiler Action: The REDEFINES clause is ignored.

Programmer Response: Probable user error. Delete the REDEFINES clause before recompiling. (Phase: DTA1)

CBL–0263 C INVALID CLAUSE IN LEVEL-66 ITEM--CLAUSE IGNORED

Explanation: A clause other than RENAMES was specified for a level-66 data item.

Compiler Action: The invalid clause is ignored.

Programmer Response: Probable user error. Delete invalid clause from level-66 item before recompiling. (Phase: DTA1)

CBL–0264 E INVALID RENAMES CLAUSE--CLAUSE IGNORED

Explanation: The keywords RENAMES or THROUGH not followed by a data-name, the second data-name (if present) was the same as the first, or the RENAMES clause was specified for an item whose level is other than 66.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct invalid clause before recompiling. (Phase: DTA1)

CBL–0265 E NO RENAMES IN LEVEL-66 ITEM--ITEM IGNORED

Explanation: The RENAMES clause was omitted from the level-66 item.

Compiler Action: The item is ignored.

Programmer Response: Probable user error. Supply RENAMES clause before recompiling. (Phase: DTA1)

CBL–0266 E INVALID OBJECT OF RENAMES CLAUSE--ITEM IGNORED

Explanation: The object of the RENAMES clause is not an item or valid range of items within the immediately preceding record description, it describes an item whose length is variable, or the renamed items contain or are subordinate to an occurs clause or are levels 01, 66, 77, or 88.

Compiler Action: The entire level-66 item is ignored.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: DTA1)

CBL–0267 C SECTION IS MISSING IN SECTION HEADER--ACCEPTED

Explanation: The word SECTION is missing from the File, Working-Storage, or Linkage Section header.

Compiler Action: The word SECTION is assumed to be present.

Programmer Response: Probable user error. Correct section header before recompiling. (Phase: DTA1)

CBL–0268 C INVALID BLOCK CLAUSE--IGNORED

Explanation: A syntax error has been found in the BLOCK CONTAINS clause. The integer is either missing, misplaced, too large, or specified for a TRANSACTION file.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct syntax of clause before recompiling. (Phase: DTA3)

CBL–0269 C INVALID RECORD CLAUSE--IGNORED

Explanation: A syntax error has been found in the RECORD CONTAINS clause. The integer is either missing or misplaced.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct syntax of clause before recompiling. (Phase: DTA3)

CBL–0270 C LABEL CLAUSE FOR FILE INVALID OR MISSING

Explanation: OMITTED labels were specified for a disk file, STANDARD labels were specified for a TRANSACTION file, or the LABEL RECORDS clause was omitted for the FD entry.

Compiler Action: The file-name is processed as if the clause were specified correctly.

Programmer Response: Probable user error. Correct the clause before recompiling. (Phase: DTA3)

CBL–0271 C INVALID ASCENDING/DESCENDING KEY PHRASE IN OCCURS CLAUSE--PHRASE IGNORED

Explanation: No data name was found, a qualifier connective (IN or OF) not followed by a data name, or an invalid key has been specified. The first key must be either the subject of the OCCURS clause or subordinate to the subject; all following keys must be subordinate to the subject. Any key that is not the subject of the OCCURS clause must not contain or be subordinate to another OCCURS clause.

Compiler Action: The ASCENDING/DESCENDING phrase is ignored.

Programmer Response: Probable user error. Correct the invalid phrase before recompiling. (Phase: DTA1, DLIT)

CBL–0273 C DUPLICATE CLAUSE IN FILE DESCRIPTION--FIRST ONE USED

Explanation: More than one specification for a clause has been found in the file description entry.

Compiler Action: The first clause is used.

Programmer Response: Probable user error. Delete duplication before recompiling. (Phase: DTA3)

CBL–0280 E PRECEDING ITEM/name/REDEFINES A LARGER ITEM

Explanation: The indicated data item (other than an 01 item) redefines a larger item.

Compiler Action: The difference in the sizes of the two items is allocated following the smaller item. (Both the redefined and redefining fields will have the same left address.)

Programmer Response: Probable user error. Increase the size of the smaller item before recompiling. (Phase: DTA1)

CBL–0285 E NO VALID SECTION PRECEDING DATA ITEM--WORKING STORAGE ASSUMED

Explanation: A level item was encountered immediately following the Data Division header.

Compiler Action: The item is assumed to be in the Working-Storage Section.

Programmer Response: Probable user error. Provide a valid section header before recompiling. (Phase: DTA1)

CBL–0286 E DATA ITEM AFTER FD OR SD NOT LEVEL 01--FILE IGNORED

Explanation: The first data-item following an FD or SD entry must have level-01.

Compiler Action: The file is ignored.

Programmer Response: Probable user error. Ensure that the FD or SD entry is followed by a level-01 record description before recompiling. (Phase: DTA1)

CBL–0287 E REQUIRED KEYWORD(S) MISSING FROM SIGN CLAUSE--CLAUSE IGNORED

Explanation: The SIGN clause did not contain the words LEADING or TRAILING, or SEPARATE was not found with the keyword CHARACTER.

Compiler Action: The SIGN clause is ignored.

Programmer Response: Probable user error. Correct the SIGN clause before recompiling. (Phase: DTA1)

CBL–0288 E SIGN CLAUSE FOR NON-NUMERIC DISPLAY ELEMENTARY ITEM--CLAUSE IGNORED

Explanation: A SIGN clause is specified for an elementary item that is not a numeric display item.

Compiler Action: The SIGN clause is ignored.

Programmer Response: Probable user error. Specify USAGE IS DISPLAY before recompiling. (Phase: DTA1)

CBL–0289 E SIGN CLAUSE INVALID FOR ITEM WITH UNSIGNED PICTURE--SIGN CLAUSE IGNORED

Explanation: The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character S or for a group item containing at least one such numeric data description entry.

Compiler Action: The SIGN clause is ignored.

Programmer Response: Probable user error. Correct the PICTURE or SIGN clause before recompiling. (Phase: DTA1)

CBL–0290 E SIGN CLAUSE IN ELEMENTARY ITEM DIFFERS FROM SIGN OF GROUP--GROUP SIGN USED

Explanation: If a group item contains a SIGN clause, any SIGN clauses within the group must be the same as the group SIGN clause. The level items in the group do not have to have a SIGN clause.

Compiler Action: The group sign is used.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: DTA1)

CBL–0292 C INVALID INTEGER OR DATA-NAME IN LINAGE CLAUSE

Explanation: The LINAGE value integer was equal to zero. If the name of a variable was specified instead of a numeric literal, the variable is either not in the DATA DIVISION or is of an invalid type.

Compiler Action: The compiler allows this value to default to the system value for the number of lines per page at execution time.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: DTA3)

CBL–0293 C LINAGE CLAUSE SPECIFIED FOR INVALID DEVICE--IGNORED

Explanation: The LINAGE clause can only be specified for a printer.

Compiler Action: The clause is ignored.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: DTA3)

CBL–0295 W PICTURE UNSIGNED AND VALUE SIGNED--SIGN IGNORED

Explanation: The literal following the VALUE clause should not be signed if the PICTURE is unsigned.

Compiler Action: The sign is ignored.

Programmer Response: Probable user error. Correct invalid sign before recompiling. (Phase: DLIT)

CBL–0296 C INVALID INTEGER OR DATA-NAME IN FOOTING CLAUSE

Explanation: The FOOTING clause has a syntax error, the integer value specified is greater than the LINAGE value, or the value is equal to zero. If the name of a variable was specified instead of a numeric literal, the variable is either not in the DATA DIVISION or is of an invalid type.

Compiler Action: The FOOTING value is assumed to be equal to the LINAGE value.

Programmer Response: Probable user error. Correct the invalid specification before recompiling. (Phase: DTA3)

CBL–0297 C INVALID INTEGER OR DATA-NAME IN TOP CLAUSE

Explanation: The TOP clause has a syntax error. If the name of a variable was specified instead of a numeric literal, the variable is either not in the DATA DIVISION or is of an invalid type.

Compiler Action: The TOP value is assumed to be zero.

Programmer Response: Probable user error. Correct the invalid specification before recompiling. (Phase: DTA3)

CBL–0298 C INVALID INTEGER OR DATA-NAME IN BOTTOM CLAUSE

Explanation: The BOTTOM clause has a syntax error. If the name of a variable was specified instead of a numeric literal, the variable is either not in the DATA DIVISION or is of an invalid type.

Compiler Action: The BOTTOM value is assumed to be zero.

Programmer Response: Probable user error. Correct the invalid specification before recompiling. (Phase: DTA3)

CBL–0300 E /name/IS DEFINED AS BOTH PARAGRAPH AND SECTION

Explanation: The indicated name appears as both a paragraph and a section.

Compiler Action: The duplicate header is ignored. All procedure references to the name are made to the first procedure header.

Programmer Response: Probable user error. Correct or delete the indicated name before recompiling. (Phase: PRO2)

CBL–0301 E DUPLICATE SECTION/name/

Explanation: The indicated section-name element is duplicated in the program.

Compiler Action: The duplicate section header is ignored. All references to the name will be resolved to the first appearance of the section header.

Programmer Response: Probable user error. Correct or delete indicated name before recompiling. (Phase: PRO2)

CBL–0302 E DUPLICATE PARAGRAPH/name/IN SECTION

Explanation: Two or more paragraphs within the same Procedure Division section have the same name.

Compiler Action: The duplicate paragraph-name is ignored. All references to the name will be resolved to the first appearance of the paragraph.

Programmer Response: Probable user error. Correct or delete indicated name before recompiling. (Phase: PRO2)

CBL–0303 E INVALID QUALIFICATION OF/name or keyword/

Explanation: The indicated name is improperly qualified.

Compiler Action: The compiler substitutes a dummy item for the invalid term, and processing continues. However, incorrect code will be generated for the statement.

Programmer Response: Probable user error. Correct indicated element before recompiling. (Phase: DTA1, DTA2, DLIT, PRO1, PROA, PRO3)

CBL–0304 E INSUFFICIENT OR MISSING QUALIFICATION OF/name or keyword/

Explanation: The indicated name is not unique.

Compiler Action: The compiler substitutes a dummy item for the invalid item, and processing continues. However, incorrect code will be generated for the statement.

Programmer Response: Probable user error. Supply the missing qualification for the indicated element before recompiling. (Phase: DTA2, DLIT, PRO1, PROA, PRO3)

CBL–0305 E LEVEL OF QUALIFICATION EXCEEDS LIMIT

Explanation: Only 51 levels of qualification are allowed for a data-name, and only 2 levels are allowed for a procedure-name.

Compiler Action: The compiler substitutes a dummy item for the invalid item, and processing continues. However, invalid code will be generated for the statement.

Programmer Response: Probable user error. Before recompiling, ensure that level of qualifications does not exceed the limit. (Phase: DTA2, DLIT, PRO1, PROA)

CBL–0306 E /name/IS INVALID PARAGRAPH TO ALTER

Explanation: A paragraph referenced by an ALTER statement must contain only a single GO TO statement.

Compiler Action: The ALTER statement is not generated.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: PRO3)

CBL–0307 C PERIOD REQUIRED--PERIOD ASSUMED

Explanation: The last statement preceding a paragraph does not end with a period.

Compiler Action: The period is assumed to be present.

Programmer Response: Probable user error. Supply required period before recompiling. (Phase: PRO1, PROA, PRO2)

CBL–0308 W PROCEDURE DIVISION HEADER ERROR

Explanation: The Procedure Division must begin with the header PROCEDURE DIVISION followed by a period or a USING option. This error also occurs if the statement following the Procedure Division header is invalid.

Compiler Action: The keywords PROCEDURE and DIVISION are assumed to be correct. The USING option, if present, is processed as if the two keywords were correct.

Programmer Response: Probable user error. Correct error in header before recompiling. (Phase: PRO1)

CBL–0309 W EXIT MUST BE ONLY STATEMENT IN PARAGRAPH

Explanation: A paragraph contains one or more statements in addition to the EXIT statement.

Compiler Action: Code is generated as if the EXIT statement were correct.

Programmer Response: Probable user error. Correct error in statement before recompiling. (Phase: PRO2)

CBL–0310 E TOO MANY SUBSCRIPTS OR INDEXES SPECIFIED FOR/name/

Explanation: The number of subscript or index specifications exceeds the level of OCCURS of the name.

Compiler Action: The subscript or index values are assumed to be 1.

Programmer Response: Probable user error. Correct subscript or index specification before recompiling. (Phase: PROA)

CBL–0311 E TOO FEW SUBSCRIPTS OR INDEXES SPECIFIED FOR/name or keyword/

Explanation: The level of OCCURS for the name exceeds the number of subscripts or indexes specified.

Compiler Action: The subscript or index values are assumed to be 1.

Programmer Response: Probable user error. Correct subscript or index specification before recompiling. (Phase: PROA)

CBL–0313 C MISSING RIGHT PARENTHESIS--STATEMENT IGNORED

Explanation: Each left parenthesis must be paired with a right parenthesis.

Compiler Action: The statement is dropped with no further syntax checking.

Programmer Response: Probable user error. Supply missing parenthesis before recompiling. (Phase: PRO3)

CBL–0314 E LINKAGE SECTION NAME/name/NOT IN USING STATEMENT--IGNORED

Explanation: The indicated item is level-01 (or level-77) or is subordinate to a level-01 data description in the Linkage Section whose name does not appear in the USING option of the Procedure Division header.

Compiler Action: The compiler substitutes a dummy item for the invalid item, and processing continues. However, incorrect code will be generated for the statement.

Programmer Response: Probable user error. Correct the USING option before recompiling. (Phase: PROA)

CBL–0315 E /name or keyword/INVALID INDEXED OR SUBSCRIPTED ITEM

Explanation: Either the indicated element is not a data-name or its data description does not have an OCCURS clause.

Compiler Action: The expression within parentheses following the element is discarded.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: PROA)

CBL–0316 E /name or literal/NOT NUMERIC--FOUND IN ARITHMETIC EXPRESSION

Explanation: The indicated element is neither a numeric elementary item nor a numeric literal.

Compiler Action: The expression is ignored.

Programmer Response: Probable user error. Replace the indicated element with a numeric item before recompiling. (Phase: PRO3)

CBL–0320 E MISSING LEFT PARENTHESIS--STATEMENT IGNORED

Explanation: A right parenthesis has been found without a corresponding left parenthesis.

Compiler Action: The statement is dropped with no further syntax checking.

Programmer Response: Probable user error. Supply missing parenthesis before recompiling. (Phase: PRO3)

CBL–0322 E /name/ NOT A GROUP ITEM--STATEMENT IGNORED

Explanation: Each identifier in a ADD/SUBTRACT/MOVE statement with the CORRESPONDING option must identify a group item.

Compiler Action: The statement is dropped with no further processing.

Programmer Response: Probable user error. Ensure that both items are group items. (Phase: PRO1)

CBL–0323 W NO CORRESPONDING ITEMS FOUND--STATEMENT IGNORED

Explanation: No corresponding items were found for the group items specified in a ADD/SUBTRACT/MOVE statement with the CORRESPONDING option.

Compiler Action: The statement is dropped.

Programmer Response: Probable user error. Ensure that there is at least one corresponding item pair. (Phase PRO1)

CBL–0324 E INVALID OR MISSING KEYWORDS IN ALTER--TO ASSUMED

Explanation: An ALTER statement has been found with a missing or incorrect keyword(s) following the first procedure-name of a pair.

Compiler Action: The keyword TO is assumed and replaces all keywords present.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: PRO2)

CBL–0325 E /name or keyword/INVALID AT THIS POINT

Explanation: An unexpected element was encountered.

Compiler Action: The compiler ignores the invalid element and any associated operands. Incorrect code will be generated for the statement.

Programmer Response: Probable user error. Correct indicated item before recompiling. (Phase: PRO1, PROA, PRO2, PRO3)

CBL–0326 E /insert/IS NOT A DEFINED NAME

Explanation: An undefined name has been found.

Compiler Action: The compiler substitutes a dummy item for the undefined name, and processing continues. However, incorrect code will be generated for the statement.

Programmer Response: Probable user error. Supply a valid name before recompiling. (Phase: PRO3)

CBL–0328 C MISSING OR INVALID LITERAL IN INDEX SPECIFICATION--FIRST OCCURRENCE USED

Explanation: An index specification contains a plus or minus sign that is not followed by a valid numeric literal.

Compiler Action: All index values are assumed to be 1.

Programmer Response: Probable user error. Correct invalid literal before recompiling. (Phase: PROA)

CBL–0329 E SUBSCRIPT OR INDEX LITERAL NOT INTEGER

Explanation: A literal was not a positive integer in one of the following cases:

- Within an index or subscript specification
- Within a SET statement
- Within a VARYING option of a PERFORM statement that references an index-name
- Within a relational condition referencing an index-name

Compiler Action: If the error occurred within an index or subscript specification, all indexes or subscripts are assumed to have a value of 1. If it occurred elsewhere, a value of 1 is substituted for the invalid literal.

Programmer Response: Probable user error. Ensure that the literal is a positive integer before recompiling. (Phase: PROA)

CBL–0330 E INVALID LITERAL IN INDEX OR SUBSCRIPT

Explanation: The subscript or index literal either exceeds 32,768, is negative or is zero in one of the following cases:

- Within an index or subscript specification
- Within a SET statement
- Within a VARYING option of a PERFORM statement that references an index-name
- Within a relational condition referencing an index-name

Compiler Action: If the error occurred within an index or subscript specification, all subscripts or indexes are assumed to have a value of 1. If it occurred elsewhere and if the literal was greater than 32,768, a value of 1 is substituted. If it occurred elsewhere and the literal was negative, its absolute value is substituted.

Programmer Response: Probable user error. Correct invalid literal before recompiling. (Phase: PROA, GEN2)

CBL–0331 E /insert/IS NOT DEFINED IN THE DATA DIVISION

Explanation: The identifier was not defined.

Compiler Action: The compiler substitutes a dummy item for the invalid item, and processing continues. However, incorrect code will be generated for the statement.

Programmer Response: Probable user error. Supply valid name before recompiling. (Phase: PROA, PRO3)

CBL–0332 E /name or keyword/IS NOT A VALID SUBSCRIPT

Explanation: The identifier used as a subscript is either a nonnumeric data-name or it requires subscripting itself.

Compiler Action: All subscripts or indexes are assumed to have a value of 1.

Programmer Response: Probable user error. Supply subscript item before recompiling. (Phase: PROA)

CBL–0333 E NAME NOT DEFINED AS INDEXED BY/name/

Explanation: The indicated name and index belong to different tables. A name may be indexed only by index-names of the table to which it belongs. In addition, the index-names must be in the proper order within the index specifications.

Compiler Action: The value of all subscripts and indexes is assumed to be 1.

Programmer Response: Probable user error. Correct invalid index specification before recompiling. (Phase: PROA)

CBL–0334 E NO INDEX DEFINED FOR TABLE IN SEARCH--STATEMENT IGNORED

Explanation: The Data Division description of the table to be searched must contain the INDEXED BY option.

Compiler Action: The statement is dropped.

Programmer Response: Probable user error. Ensure that the Data Division description of the table to be searched contains the INDEXED BY option. (Phase: PRO1)

CBL–0337 C TABLE/name/IN SEARCH STATEMENT HAS SUBSCRIPT/INDEX--SUBSCRIPT/INDEX IGNORED.

Explanation: The name of the table to be searched must not be subscripted or indexed.

Compiler Action: The subscript/index(s) is dropped.

Programmer Response: Probable user error. Ensure that the table specified in the SEARCH statement is not subscripted or indexed. (Phase: PROA)

CBL–0338 E /name/IS BOTH DATA-NAME AND PROCEDURE-NAME

Explanation: The indicated name is defined both in the Data Division and Procedure Division.

Compiler Action: The compiler substitutes a dummy item for the invalid item, and processing continues. However, incorrect code will be generated for the statement.

Programmer Response: Probable user error. Correct indicated item before recompiling. (Phase: PRO2, PRO3)

CBL–0339 C /name/CONTAINS AMBIGUOUS INDICATOR ASSOCIATION VALUES

Explanation: The INDICATOR data-item specified in the WRITE statement is a group item that contains an indicator definition subordinate to an item which contains an OCCURS clause. For example:

```
01   INDIC-LIST.
     05   INDIC-5 PIC   1 INDICATOR 5.
     05   A-TABLE  OCCURS  10 TIMES.
          10   FILLER PIC X(5).
          10   INDIC-10  PIC  1  INDICATOR 10.
```

Compiler Action: The INDICATOR data-item is accepted. Indicators defined subordinate to an item with an OCCURS are assigned to their first occurrence in the table.

Programmer Response: Probable user error. Ensure that any table subordinate to the INDICATOR data-item does not contain mixed indicator and nonindicator elementary items. (Phase: PRO1)

CBL–0340 E ALL FOLLOWED BY INVALID LITERAL

Explanation: The literal following ALL must be either a nonnumeric literal or a figurative constant.

Compiler Action: If the literal is numeric, the ALL is dropped.

Programmer Response: Probable user error. Correct invalid literal before recompiling. (Phase: PROA)

CBL–0341 E INVALID 'ALL' LITERAL--'ALL' IGNORED

Explanation: The figurative constant 'ALL literal' is invalid in a DISPLAY, STRING, or STOP statement.

Compiler Action: The keyword 'ALL' is dropped.

Programmer Response: Probable user error. Ensure that the figurative constant 'ALL literal' is not specified in a DISPLAY, STRING or STOP statement. (Phase: PROA)

CBL–0342 C ILLEGAL OPERATOR IN SEARCH ALL...WHEN PHRASE--ACCEPTED

Explanation: The conditional operators EQUAL and AND are the only conditional operators permitted in the WHEN phrase of a format 2 SEARCH statement.

Compiler Action: Code is generated as if the illegal operators were legal.

Programmer Action: Probable user error. Ensure that the WHEN phrase of the format 2 SEARCH statement contains no conditional operators except EQUAL and AND. (Phase: PROA)

CBL–0343 C MULTIPLE VALUE CONDITION-NAME IN SEARCH ALL...WHEN PHRASE--ACCEPTED

Explanation: In Format 2 of the SEARCH statement, condition-names must be defined as having only a single value.

Compiler Action: The multiple value condition-name is accepted. Code is generated to test the multiple values.

Programmer Response: Probable user error. Ensure that all referenced condition-names are defined as having only a single value. (Phase: PROA)

CBL-0344 E INVALID NAME OR LITERAL ENTRY IN USING CLAUSE

Explanation: A parameter following the USING option either was not a name, was a name that did not appear in the Linkage Section, or was not a 01 or 77 level item for a Procedure Division header.

Compiler Action: If the parameter was not a name, the remainder of the USING option is ignored. If the name did not appear in the Linkage Section, or if the name was not the right level item, that name is ignored.

Programmer Response: Probable user error. Correct invalid entry before recompiling. (Phase: PRO1)

CBL-0345 C NO SECTION-NAME AFTER PROCEDURE DIVISION HEADER--ACCEPTED

Explanation: If sections are used within the Procedure Division, a section header must immediately follow the Procedure Division header.

Compiler Action: None.

Programmer Response: Probable user error. Before recompiling ensure that the section header precedes the first paragraph of the Procedure Division. (Phase: PRO2)

CBL-0347 E NUMBER OF LINKAGE NAMES EXCEEDS 15

Explanation: A maximum of 15 parameters may appear in the USING option.

Compiler Action: All parameters after the first 15 are ignored.

Programming Response: Probable user error. Before recompiling, ensure that not more than 15 parameters appear. (Phase: PRO1)

CBL-0348 E INVALID ELEMENT AFTER PROCEDURE HEADER--IGNORED

Explanation: Extraneous elements have been found following a paragraph or section header.

Compiler Action: The extraneous element or elements are ignored. Processing continues at the next statement.

Programmer Response: Probable user error. Correct paragraph header, section header, or incorrect elements before recompiling. (Phase: PRO2)

CBL-0349 E DATA-NAME OR LITERAL EXPECTED--FOUND/insert/

Explanation: The indicated element was found where a data-name or literal was required, either in a subscript or index specification or in a SET statement.

Compiler Action: If the error occurred inside the parentheses for a subscript or index item, the value of all subscript or index items is assumed to be 1. If the error occurred in a SET statement, the compiler substitutes a dummy element, and processing continues; however, incorrect code will be generated for the statement.

Programmer Response: Probable user error. Replace indicated element with appropriate data-name or literal before recompiling. (Phase: PRO1, PROA)

CBL-0350 C INVALID OR MULTIPLE NOT(S) IN CONDITIONAL EXPRESSION--IGNORED

Explanation: The word NOT may not precede a condition if the condition itself contains a NOT.

Compiler Action: Both NOTs are ignored.

Programmer Response: Probable user error. Before recompiling, ensure that multiple NOTs do not appear in the conditional expression. (Phase: PRO3)

CBL-0351 E INVALID ABBREVIATED COMBINED RELATION CONDITION--STATEMENT IGNORED

Explanation: The compiler is unable to process the conditional expression because it is neither a complete conditional expression or a valid abbreviated combined relation condition.

Compiler Action: The statement is dropped.

Programmer Response: Probable user error. Ensure that the conditional expression is a complete conditional expression or a valid abbreviated combined relation condition. (Phase: PRO2)

CBL–0352 E MIXED INDEXES AND SUBSCRIPTS IN /name/--FIRST OCCURRENCE USED

Explanation: The indicated item is invalid because subscripting and indexing must not be used together in a single reference.

Compiler Action: The value of all subscripts or indexes is assumed to be 1.

Programmer Response: Probable user error. Before recompiling, specify either all subscripts or all indexes for the indicated item. (Phase: PROA)

CBL–0353 C INVALID OBJECT OF OCCURS...DEPENDING ON ASSOCIATED WITH /name/--MAXIMUM SIZE ASSUMED

Explanation: The object of the OCCURS...DEPENDING ON clause is invalid. (Message CBL-0230 may precede this message. See Message CBL-0230 for additional information.)

Compiler Action: The maximum length of the variable length table is assumed.

Programmer Response: Probable user error. Ensure that the description of the variable length table contains a valid OCCURS...DEPENDING ON clause. (Phase: PROA)

CBL – 0354 C NO PARAGRAPH NAME AFTER SECTION OR DIVISON HEADER--ACCEPTED

Explanation: A paragraph-name should follow section and Procedure Division header.

Compiler Action: If no section/paragraph name follows the Procedure Division header, the statements preceding the first section/paragraph name will not be executed at execution time.

Programmer Response: Probable user error. Supply paragraph-name before recompiling. (Phase: PRO2)

CBL–0355 E TOO MANY ERRORS IN STATEMENT--NOT ALL SYNTAX CHECKED

Explanation: The statement contains so many errors that syntax checking is suspended.

Compiler Action: The entire statement is ignored.

Programmer Response: Probable user error. Correct syntax of indicated statement before recompiling. (Phase: PRO1, PROA, PRO2, PRO3)

CBL–0359 C /name/IS NOT A FILE-NAME--NAME IGNORED

Explanation: The name specified in the USE ON EXCEPTION/ERROR sentence is not a file-name.

Compiler Action: The name is ignored. If more than one name was specified, the other names are processed in the normal manner.

Programmer Response: Probable user error. Ensure that all names specified in the USE ON EXCEPTION/ERROR sentence are file-names. (Phase: PRO1)

CBL–0360 E /name/SPECIFIED MORE THAN ONCE--IGNORED

Explanation: For any given file, only one EXCEPTION/ERROR procedure may be specified.

Compiler Action: The file-name is ignored.

Programmer Response: Probable user error. Ensure that for any given file, only one EXCEPTION/ERROR procedure is specified. (Phase: PRO1)

CBL–0361 E NO SECTION PRECEDING DECLARATIVE SENTENCE--SENTENCE IGNORED

Explanation: Each declarative sentence must directly follow a section header in the Declaratives.

Compiler Action: The declarative sentence is ignored.

Programmer Response: Probable user error. Ensure that each declarative sentence is directly preceded by a section header. (Phase: PRO1)

CBL-0362 C INVALID USE SENTENCE, NO CONTROLS SPECIFIED--SENTENCE IGNORED

Explanation: A required keyword or file-name was not specified in the USE ON EXCEPTION/ERROR sentence.

Compiler Action: The USE sentence is ignored.

Programmer Response: Probable user error. Ensure that either a file-name or a keyword is specified in the USE ON EXCEPTION/ERROR sentence. (Phase: PRO1, PRO3)

CBL-0363 E INVALID OR MISSING SEGMENT NUMBER--USING PREVIOUS VALID SEGMENT NUMBER

Explanation: The segment number must be an integer ranging in value from 0 through 99.

Compiler Action: The segment number of the previous section is assumed.

Programmer Response: Probable user error. Supply valid segment number before recompiling. (Phase: PRO2)

CBL-0366 W SEGMENT NUMBER SIGNED--SIGN IGNORED

Explanation: The segment number must be unsigned.

Compiler Action: The segment number is assumed to be unsigned.

Programmer Response: Probable user error. Provide an unsigned segment number before recompiling. (Phase: PRO2)

CBL-0367 W SUBPROGRAM MAY NOT BE SEGMENTED

Explanation: Only the main program may be segmented. The program being compiled is assumed to be intended as a subprogram because the Procedure Division header contains a USING option.

Compiler Action: None.

Programmer Response: Probable user error. Correct invalid specification before recompiling. If any other program to be linked is segmented, the program structure must be redesigned. If this is the only segmented program to be linked, it may be linked with its calling program without modification. (Phase: PRO2)

CBL-0368 E INVALID SYNTAX--STATEMENT DROPPED

Explanation: The compiler was attempting to process a conditional or arithmetic expression when an invalid sequence of operands or operators was encountered.

Compiler Action: The statement is ignored.

Programmer Response: Probable user error. Correct invalid syntax before recompiling. (Phase: PRO1, PROA, PRO3)

CBL-0369 W USE SENTENCE NOT ENDED WITH PERIOD SPACE--ACCEPTED

Explanation: The USE sentence must be ended with a period and a space.

Compiler Action: The missing period and space are assumed.

Programmer Response: Probable user error. Ensure that the USE sentence is followed by a period and a space. (Phase: PRO1)

CBL–0370 E LOGICAL OR ARITHMETIC EXPRESSION TOO COMPLEX

Explanation: Either a compound condition or an arithmetic expression exceeds the compiler's capacity for temporary storage.

Compiler Action: The entire statement is dropped.

Programmer Response: Probable user error. Simplify the statement either by separating it into two or more statements or by removing excessive parentheses. (Phase: PRO3)

CBL–0371 C /name/ IS INVALID AS AN INDICATOR DATA-ITEM

Explanation: An INDICATOR data-item must be an elementary Boolean data-item specified without the OCCURS clause or a group item that has elementary Boolean data-items subordinate to it.

Compiler Action: The specified INDICATOR data-item is accepted, but the indicator association values generated for this WRITE statement might be invalid.

Programmer Response: Probable user error. Ensure that the specified INDICATOR data-item is an elementary Boolean data-item without the OCCURS clause or is a group item. (Phase: PRO1)

CBL–0372 E REQUIRED KEYWORD MISSING IN 'USE' SENTENCE--SENTENCE IGNORED

Explanation: A required keyword was missing in a USE FOR DEBUGGING sentence.

Compiler Action: The USE FOR DEBUGGING sentence is ignored.

Programmer Response: Probable user error. Ensure that all required keywords are specified in the USE FOR DEBUGGING sentence. (Phase: PRO2)

CBL–0373 C SECTION/PARAGRAPH PRECEDES DECLARATIVES HEADER

Explanation: The keyword DECLARATIVES must be written on the next line after the Procedure Division header.

Compiler Action: The section/paragraph name is accepted. Any reference to this section/paragraph will produce unpredictable results.

Programmer Response: Probable user error. Ensure that the Declaratives header immediately follows the Procedure Division header. (Phase: PRO2)

CBL–0374 C PROCEDURE NAME INVALID AFTER 'ALL PROCEDURES'--'USE' SENTENCE IGNORED

Explanation: When ALL PROCEDURES is specified in a USE FOR DEBUGGING sentence, a procedure name must not be specified in any USE FOR DEBUGGING sentence.

Compiler Action: The USE FOR DEBUGGING sentence is ignored.

Program Response: Probable user error. Ensure that no procedure names are specified in any USE FOR DEBUGGING sentence if ALL PROCEDURES is specified on a USE FOR DEBUGGING sentence. (Phase: PRO3)

CBL–0375 W INVALID SEGMENT PRIORITY FOR DECLARATIVES SECTION--ZERO ASSUMED

Explanation: Sections in the Declaratives must specify a segment priority less than 50.

Compiler Action: A segment priority of zero is assumed.

Programmer Response: Probable user error. If a segment priority is specified, ensure that it is in a valid format and a value less than 50. (Phase: PRO2)

CBL–0376 C 'USE' SENTENCE DOES NOT FOLLOW SECTION HEADER

Explanation: In the Declaratives section of the Procedure Division, a USE sentence must follow a section header.

Compiler Action: None.

Programmer Response: Probable user error. Ensure that the Section header in the Declaratives is followed by a USE sentence. (Phase: PRO2)

CBL–0377 E SECTION HEADER MUST IMMEDIATELY FOLLOW 'END DECLARATIVES'

Explanation: A section header does not immediately follow the 'END DECLARATIVES.'

Compiler Action: The first Section/Paragraph name is used as the program entry point.

Programmer Response: Probable user error. Ensure that a Section header follows the 'END DECLARATIVES'. (Phase: PRO2)

CBL–0378 C PROCEDURE /name/ IS IN THE DECLARATIVES SECTION

Explanation: Procedure names within debugging declarative sections must not appear in any USE FOR DEBUGGING sentence.

Compiler Action: The procedure-name is ignored.

Programmer Response: Probable user error. Ensure that procedure-names within debugging declarative sections do not appear in any USE FOR DEBUGGING sentence. (Phase: PRO3)

CBL–0379 E 'ALL PROCEDURES' INVALID AFTER PROCEDURE NAME--'USE' SENTENCE IGNORED

Explanation: When ALL PROCEDURES is specified in a USE FOR DEBUGGING sentence, procedure-names must not be specified in any USE FOR DEBUGGING sentence.

Compiler Action: The USE FOR DEBUGGING sentence is ignored.

Programmer Response: Probable user error. Ensure that ALL PROCEDURES is not specified in a USE FOR DEBUGGING sentence after procedure-name has been specified in A USE FOR DEBUGGING sentence. (Phase: PRO3)

CBL–0380 C PROCEDURE NAME /name/ ALREADY SPECIFIED--IGNORED

Explanation: A procedure-name may appear in only one USE FOR DEBUGGING sentence, and it may appear only once in that sentence.

Compiler Action: The procedure-name is ignored.

Programmer Response: Probable user error. Ensure that a procedure-name is specified only once in USE FOR DEBUGGING sentences. (Phase: PRO3)

CBL–0382 C ALL OTHER DECLARATIVE PROCEDURES MUST FOLLOW ALL 'USE FOR DEBUGGING' PROCEDURES--ACCEPTED

Explanation: When specified, all debugging sections must be written immediatley after the DECLARATIVES header.

Compiler Action: Normal processing.

Programmer Response: Probable user error. Ensure that all debugging sections immediately follow the DECLARATIVES header. (Phase: PRO2)

CBL–0383 E PROCEDURE NAME /name/ IS AN INVALID PROCEDURE REFERENCE FOR THIS SECTION

Explanation: Procedure references must conform to the following rules:

Within a Declarative procedure, there must be no reference to any nondeclarative procedure. In the nondeclarative portion of the program, there must be no reference to procedure-names that appear in an EXCEPTION/ERROR Declarative procedure, except that PERFORM statements may refer to an EXCEPTION/ERROR procedure or to procedures associated with it.

Statements appearing outside the debugging sections must not refer to procedure-names defined within the debugging sections.

Except for the USE FOR DEBUGGING sentence itself, statements within a debugging declarative section may refer to procedure-names defined in a different USE procedure only through the PERFORM statement.

Compiler Action: A null procedure reference is generated.

Programmer Response: Probable user error. Ensure that all procedure references conform to the above rules. (Phase: PRO3)

CBL–0384 E PROCEDURE-1 AND PROCEDURE-2 NOT IN THE SAME DECLARATIVES SECTION

Explanation: In a PERFORM statement, when procedure-1 THROUGH procedure-2' is specified and either is a procedure in the declarative section of the program, then both procedures must be in the same declarative section.

Compiler Action: The procedure-names are accepted. Erroneous code may be generated.

Programmer Response: Probable user error. Ensure that when PERFORM specifies procedure-names contained in the declarative section of the program, both procedure-names are in the same declarative section. (Phase: PRO3)

CBL–0385 C MULTIPLE 'USE' SENTENCES FOLLOW SECTION HEADER

Explanation: Only one USE sentence should follow a section header in the declaratives section.

Compiler Action: If the section being processed is a debug declarative section, all USE sentences except the first are ignored.

If the section being processed is an EXCEPTION/ERROR declarative section, all USE sentences are accepted and processed.

Programmer Response: Probable user error. Ensure that only one USE sentences follows a section header in the declaratives section. (Phase: PRO3)

CBL–0386 E NO 'END DECLARATIVES' FOUND

Explanation: A DECLARATIVES statement was found in the source program but no END DECLARATIVES statement was detected following the DECLARATIVES.

Compiler Action: None. Incorrect code will be generated.

Programmer Response: Probable user error. Ensure an END DECLARATIVES statement exists after a DECLARATIVES statement. (Phase: PRO2)

CBL–0387 E /name/ IS AN INVALID KEY IN A SORT/MERGE STATEMENT, KEY IGNORED

Explanation: A key defined in an ASCENDING/DESCENDING key clause of a SORT statement violates one of the restrictions on how it can be defined.

Compiler Action: The key is ignored.

Programmer Response: Probable user error. Determine why the key is not valid and correct the error. (Phase: PROA)

CBL–0388 C TOTAL LENGTH OF SORT/MERGE KEYS EXCEED 256 BYTES, LAST KEY(S) IGNORED

Explanation: The compiler has a maximum of 256 bytes of storage for all keys associated with a SORT/MERGE statement. This can include some bytes used internally by the SORT/MERGE utility. This SORT/MERGE statement exceeded that limit.

Compiler Action: The remaining keys from the point at which the key length exceeded 256 are ignored.

Programmer Response: Probable user error. Reduce the number and/or size of the keys specified for the SORT/MERGE statement. (Phase: PROA)

CBL–0389 E NO VALID ASCENDING/DESCENDING KEYS FOUND FOR A SORT/MERGE STATEMENT

Explanation: The compiler could find no valid keys specified for a SORT/MERGE statement.

Compiler Action: The compiler will generate a call to SORT with no keys defined.

Programmer Response: Probable user error. Ensure that there is at least one valid key specified for a SORT/MERGE statement. (Phase: PROA)

CBL–0390 E SORT VERB ENCOUNTERED WITH FILE NAMED "WORK" DEFINED

Explanation: SORT uses a file named WORK to perform certain functions during execution. Because the COBOL programmer has defined a file named WORK, SORT will attempt to use the programmer's file during execution.

Compiler Action: The program will be generated. Problems will occur when SORT attempts to use the WORK file.

Programmer Response: Probable user error. Change the name of the file from WORK to another valid file name. (Phase: PRO2)

CBL–0400 E OPEN OPTION INCOMPATIBLE WITH FILE SPECIFICATION FOR/name/

Explanation: The OPEN statement is invalid for the file in question.

Compiler Action: Code is not generated for the OPEN of this file, but it is generated for all other valid file-names in the statement.

Programmer Response: Probable user error. Correct invalid OPEN statement before recompiling. (Phase: GEN1)

CBL–0401 E MISSING KEYWORD INPUT, EXTEND, OUTPUT, OR I-O IN OPEN STATEMENT

Explanation: One of the required keywords in the OPEN statement has been omitted.

Compiler Action: All input elements are discarded until one of the required keywords is found. Normal processing continues after the keyword is located.

Programmer Response: Probable user error. Supply valid keyword before recompiling. (Phase: GEN1)

CBL–0403 E START OR DELETE INCOMPATIBLE WITH FILE SPECIFICATION FOR/name/

Explanation: A START statement has been issued for a file that is not a sequentially accessed indexed or relative file, or a DELETE statement has been issued for a sequential file.

Compiler Action: No code is generated for the START statement.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN1)

CBL–0405 E READ INCOMPATIBLE WITH FILE SPECIFICATION FOR/name/

Explanation: A READ statement has been issued for a file that can only be opened for output.

Compiler Action: No code is generated for the READ statement.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN1)

CBL–0406 C INTO OPTION INVALID FOR MULTIPLE LOGICAL RECORDS OF /name/--ACCEPTED

Explanation: The INTO option is invalid for files containing multiple level-01 records of varying length in the File Section.

Compiler Action: The first record description found for the file is used.

Programmer Response: Probable user error. Correct invalid READ INTO statement before recompiling. (Phase: GEN1)

CBL–0408 E WRITE INCOMPATIBLE WITH FILE SPECIFICATION FOR/name/

Explanation: A WRITE statement has been issued for a file described as an input file.

Compiler Action: No code is generated for the WRITE statement.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN1)

CBL–0409 E /insert/INCOMPATIBLE WITH FILE SPECIFICATION FOR/name/

Explanation: Printer options have been specified for a file that is not a print file (EOP PAGE), an ADVANCING option has been specified for a file that is not unit-record output, or the mnemonic-name used in the ADVANCING option is unsuitable.

Compiler Action: No code is generated for the WRITE statement.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: GEN1)

CBL–0410 C LINAGE CLAUSE NOT SPECIFIED FOR/name/--END-OF-PAGE OPTION INVALID

Explanation: An EOP option has been specified for a file for which no LINAGE clause was specified.

Compiler Action: The EOP option is ignored. The code intended as an EOP routine will be executed after each WRITE.

Programmer Response: Probable user error. Supply required LINAGE clause before recompiling. (Phase: GEN1)

CBL–0411 E REWRITE INCOMPATIBLE WITH FILE SPECIFICATION FOR/name/

Explanation: A REWRITE statement has been issued for a file that cannot be opened in I-O mode.

Compiler Action: No code is generated for the REWRITE statement.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN1)

This page is intentionally left blank.

CBL–0412 W MORE THAN ONE USE PROCEDURE SPECIFIED FOR FILE/insert/

Explanation: The file being OPENed has an implicit USE PROCEDURE defined for INPUT, OUTPUT, I/O or EXTEND and an explicit USE PROCEDURE for this file.

*Compiler Action:*The explicit USE PROCEDURE will be used for this file.

Programmer Response: Probable user error. If the implicit USE PROCEDURE was desired, remove the explicit USE PROCEDURE for this file before recompiling. (Phase: GEN1)

CBL–0413 E INVALID SECTION-NAME SPECIFIED FOR SORT/MERGE INPUT OR OUTPUT PROCEDURE

Explanation: The name specified in the INPUT or OUTPUT PROCEDURE is not a section-name.

Compiler Action: Code generation and syntax checking are discontinued at the point where the message was issued and are resumed at the next statement. The completeness of generated code is unpredictable.

Programmer Response: Probable user error. Correct invalid section-name before re-compiling. (Phase: GEN1)

CBL–0414 E NUMBER OF KEYS IN A SORT OR MERGE STATEMENT EXCEEDS 12

Explanation: A thirteenth ASCENDING/DESCENDING key has been found.

Compiler Action: All keys beyond the twelfth are discarded without syntax check. Code is generated for the SORT or MERGE statement. Only the first twelve keys are used.

Programmer Response: Probable user error. Ensure that no more than twelve keys are specified before recompiling. (Phase: GEN1)

CBL–0415 E NUMBER OF FILES IN A SORT OR MERGE USING CLAUSE EXCEEDS 8

Explanation: A ninth file has been found after the USING.

Compiler Action: All files beyond the eighth file are discarded without syntax check. Code is generated for the USING clause and the first eight files are used.

Programmer Response: Probable user error. Ensure that no more than eight files appear after USING before recompiling. (Phase: GEN1)

CBL–0416 W /name/ EXCEEDS 256 CHARACTERS--TRUNCATED FOR COMPARISON

Explanation: The length of the identifier exceeds the 256-character limit of comparison for EXHIBIT CHANGED.

Compiler Action: Only the leftmost 256 characters are analyzed to determine whether the value has changed.

Programmer Response: Check that useful information is not being discarded by the truncation. (Phase: GEN2)

CBL–0418 E NUMBER OF PROCEDURE-NAMES IN GO TO DEPENDING ON STATEMENT EXCEEDS 99

Explanation: The number of procedure-names in the GO TO . . . DEPENDING ON statement is greater than 99.

Compiler Action: The first 99 procedure-names are used.

Programmer Response: Probable user error. Before recompiling, ensure that the number of procedure-names in the GO TO . . . DEPENDING ON statement does not exceed 99. (Phase: GENA)

CBL–0419 E /name/AND/name/ARE IN DIFFERENT SEGMENTS--CANNOT ALTER

Explanation: An ALTER statement has been issued for a GO TO statement in an independent segment from a segment of different priority.

Compiler Action: No code is generated.

Programmer Response: Probable user error. Correct statement for ALTER, or revise priority numbers before recompiling. (Phase: GENA)

CBL–0420 W NUMBER OF DIGITS IN/name/EXCEEDS 4--TRUNCATED

Explanation: The PICTURE for the integer in the GO TO . . . DEPENDING ON statement has more than the allowable number of 9s.

Compiler Action: The four low-order digits are used.

Programmer Response: Probable user error. Ensure that the high-order positions contain no significant digits, or specify the PICTURE for the identifier again. (Phase: GENA)

CBL–0421 E /name/AND/name/ARE IN DIFFERENT SEGMENTS--CANNOT PERFORM

Explanation: The PERFORM range includes segments of unequal priority, or a PERFORM statement in an independent segment refers to a paragraph in a different independent segment.

Compiler Action: No code is generated.

Programmer Response: Probable user error. Correct invalid statement, or revise priority numbers before recompiling. (Phase: GENA)

CBL–0422 E DEPTH OF VARYING EXCEEDS 3

Explanation: A third AFTER option has been found in a PERFORM statement.

Compiler Action: The third AFTER option and the remainder of the statement are ignored. Code is generated for the statement up to the third AFTER.

Programmer Response: Probable user error. Delete third AFTER option before recompiling. (Phase: GENA)

CBL–0423 E /insert/IS INVALID CALL PARAMETER

Explanation: A file-name, procedure-name, keyword, literal, or index has been found after USING.

Compiler Action: The invalid parameter is counted in the check for maximum-15 CALL parameters, and an address of 0000 is entered for it in the object-time parameter list.

Programmer Response: Probable user error. Supply valid CALL parameter before recompiling. (Phase: GEN2)

CBL–0424 E NUMBER OF PARAMETERS IN CALL STATEMENT EXCEEDS 15

Explanation: A sixteenth parameter has been found following USING.

Compiler Action: All parameters beyond the fifteenth are discarded without syntax check. Code is generated for the CALL and for the first 15 parameters.

Programmer Response: Probable user error. Ensure that no more than 15 parameters appear after the USING before recompiling. (Phase: GEN2)

CBL–0427 E INVALID MOVE STATEMENT--/name or literal/AND/name or literal/INCOMPATIBLE

Explanation: The sending and receiving fields in the MOVE statement are incompatible.

Compiler Action: The MOVE is not generated.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN2)

CBL–0428 E INVALID INDEX MODIFICATION--/name or literal/AND/name or literal/INCOMPATIBLE

Explanation: The two operands of the SET statement do not form a valid pair (for example, SET index UP BY index).

Note: This message may also be issued for a PERFORM VARYING statement, because SET is implied when the variable is an index.

Compiler Action: The index modification in the SET or PERFORM statement is not generated.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN2)

CBL–0429 E INVALID RELATIONAL TEST--/name or literal/AND/name or literal/INCOMPATIBLE

Explanation: An invalid comparison has been made. For instance, one of the following comparisons may have been made: arithmetic expression with alphanumeric data-name, literal with literal, index data item with binary data-name, and so on.

Compiler Action: No code is generated for the comparison, or within a compound condition, one of the fields may arbitrarily be compared with itself or to zero.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN3)

CBL–0430 W INVALID SIGN TEST OF/name/--ACCEPTED

Explanation: A NEGATIVE condition has been specified for an unsigned numeric field.

Compiler Action: The test is generated.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: GEN3)

CBL–0431 E INVALID SIGN TEST OF/name/--IGNORED

Explanation: A sign test has been applied to a nonnumeric field.

Compiler Action: No code is generated except within a compound condition for which arbitrary code may be generated.

Programmer Response: Probable user error. Ensure that indicated element is numeric before recompiling. (Phase: GEN3)

CBL–0432 E INVALID CLASS TEST OF /name/

Explanation: Either a numeric test has been requested for an alphanumeric or computational field, or an alphabetic test has been requested for a numeric field.

Compiler Action: Code generation and syntax checking are discontinued at the point where the message was issued and are resumed at the next statement (the true-condition routine). The results are unpredictable.

Programmer Response: Probable user error. Correct invalid specification before recompiling. (Phase: GEN3, PRO2)

CBL–0433 C CONDITIONAL STATEMENT INVALID AT THIS POINT--ACCEPTED

Explanation: A conditional statement has been found in the same sentence as another conditional statement.

Compiler Action: The indicated conditional statement is accepted as written. The second conditional statement will be treated as nested within the first.

Programmer Response: Probable user error. Ensure that no more than one conditional statement is specified in a sentence before recompiling. (Phase: GEN1)

CBL–0434 E LOGICAL EXPRESSION TOO COMPLEX

Explanation: Too many parentheses have been used. The level of parentheses in a compound condition exceeds the compiler's capacity for temporary storage.

Compiler Action: Code generation and syntax checking are discontinued at the point where the message was issued and are resumed at the next statement (the true-condition routine). The results are unpredictable.

Programmer Response: Probable user error. Make the compound condition a series of simple conditions before recompiling. (Phase: GEN3)

CBL–0435 W DECIMAL ALIGNMENT CAUSES /name or literal/ TO BE TREATED AS ZERO

Explanation: The difference in scaling between two numeric fields in a MOVE or arithmetic statement is such that all significant digits of one of the two fields are truncated.

Compiler Action: All fields are aligned on the decimal point.

Programmer Response: Probable user error. Check length and scaling of all elements in the statement before recompiling. (Phase: GEN2, GEN3)

CBL–0436 W HIGH-ORDER TRUNCATION MAY OCCUR

Explanation: Overflow of high-order digits may cause significance to be lost. Overflow occurs when either an intermediate result field or the final result field is too small to hold possible computed values. (For more information on intermediate result fields, see Appendix D.)

Compiler Action: None.

Programmer Response: Probable user error. Ensure that truncation will not produce incorrect results. (Phase: GEN2, GEN3)

CBL–0437 W RESULT DOES NOT HAVE MORE DECIMALS THAN RECEIVING FIELD--ROUNDING IGNORED

Explanation: The size of a fractional result does not exceed the number of places provided for its storage. (For example, ADD PIC 99V9 to PIC 99V9 ROUNDED.)

Compiler Action: No rounding takes place.

Programmer Response: Probable user error. Correct specification before recompiling. (Phase: GEN3)

CBL–0438 C /insert/INVALID--IGNORED

Explanation: An extraneous element has been found in an otherwise valid statement.

Compiler Action: The element is dropped, and processing continues.

Programmer Response: Probable user error. Delete the extraneous element before recompiling. (Phase: GEN1, GENA, GEN2, GEN3)

CBL–0439 C /insert/MISSING--ASSUMED

Explanation: A required keyword has been omitted.

Compiler Action: The missing keyword is assumed to be present.

Programmer Response: Probable user error. Supply the required keyword before recompiling. (Phase: GEN1, GENA, GEN2, GEN3)

CBL–0440 E OPTION MISSING--STATEMENT IGNORED

Explanation: One of several keywords is required for a decision as to the meaning of some statements. If such a keyword is missing, the compiler is unable to make any assumptions.

Compiler Action: No code is generated for the statement.

Programmer Response: Probable user error. Supply missing option before recompiling. (Phase: GEN1, GENA, GEN2, GEN3)

CBL–0441 E /insert/EXPECTED--/insert/ FOUND--REST OF STATEMENT IGNORED

Explanation: A source element is completely unrecognized in the context of its statement.

Compiler Action: Code generation and syntax checking are discontinued at the point where the message is issued and are resumed at the next statement.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN1, GENA, GEN2, GEN3)

CBL–0442 E /insert/INVALID RECEIVING FIELD

Explanation: A literal, procedure-name, or file-name has been specified as a receiving field.

Compiler Action: The data in the sending field of the MOVE statement or the result of an arithmetic operation will not be stored.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN2, GEN3)

CBL–0443 E /insert/INVALID HERE--NOT NUMERIC

Explanation: A nonnumeric field has been specified as an operand for an arithmetic statement.

Compiler Action: Code is not generated or, in some cases, zero is substituted for the operand.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN3)

CBL–0444 C /name or literal/INVALID HERE--NOT AN INTEGER

Explanation: An exponent used in an arithmetic expression must be an integer. An integer is required for the GO TO . . . DEPENDING ON and the PERFORM . . . TIMES statements. The element specified contains one or more positions to the right of the decimal point.

Compiler Action: The decimal positions are truncated from the exponent. In all other cases, the indicated element is assumed to have a value of zero.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GENA, GEN2, GEN3)

CBL–0445 C NAMED CHANGED SHOULD BE CHANGED NAMED--ACCEPTED

Explanation: The required order in an EXHIBIT statement is EXHIBIT [CHANGED] NAMED identifer.

Compiler Action: The statement is accepted.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN2)

CBL–0446 E /name or literal/INVALID IN THIS CONTEXT

Explanation: The operand does not fulfill a special requirement for the statement in question.

Compiler Action: No code is generated.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN1, GENA, GEN2, GEN3)

CBL–0447 E STATEMENT INCOMPLETE

Explanation: No statement appears between a test condition and the end of the statement (for example, IF A = B); or a statement ends before all required elements have been found.

Compiler Action: For an incomplete condition, the next sequential statement is executed whether the condition is satisfied or not. For other incomplete statements, results are unpredictable.

Programmer Response: Probable user error. Supply the missing statement before recompiling. (Phase: GEN1, GENA, GEN2, GEN3)

CBL–0448 E ELSE IS UNMATCHED BY IF--IGNORED

Explanation: Each ELSE must be matched by an IF statement.

Compiler Action: The unmatched ELSE is ignored, and the code is generated for the statement following the ELSE.

Programmer Response: Probable user error. Supply the missing IF statement, or delete the unmatched ELSE before recompiling. (Phase: GEN3)

CBL–0449 C NEXT SENTENCE CLAUSE INVALID HERE--ACCEPTED

Explanation: NEXT SENTENCE follows the INVALID KEY option or AT END.

Compiler Action: Invalid key or end of file causes a branch to NEXT SENTENCE.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN1)

CBL–0450 W STATEMENT WILL NOT BE EXECUTED

Explanation: A statement follows a GO TO or STOP RUN statement without an intervening period or ELSE statement.

Compiler Action: Correct code is generated.

Programmer Response: Probable user error. Insert period or ELSE where required by program logic. (Phase: GEN1, GENA, GEN2, GEN3)

CBL–0452 E ASSUMING/insert/STANDS FOR/insert/

Explanation: A source element is invalid where it appears; however, the following element is valid, indicating that perhaps the invalid element is the result of a keying error.

Compiler Action: The necessary keyword is assumed, and the erroneous element is dropped.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN1, GENA, GEN2, GEN3)

CBL–0453 C MAXIMUM NUMBER OF COMPARISON OPERANDS EXCEEDED--EXCESS IGNORED

Explanation: No more than 15 comparison operands may be specified for REPLACING option or a TALLYING option.

Compiler Action: All comparison operands after the first 15 are ignored.

Programmer Response: Probable user error. Ensure that the REPLACING option or TALLYING option contains no more than 15 comparison operands before recompiling. (Phase: GEN2)

CBL–0454 E OPEN OPTION INCOMPATIBLE WITH RESERVE CLAUSE FOR/file-name/--IGNORED

Explanation: OPEN I-O or INPUT may not be issued for an indexed sequential file that is specified with RESERVE 2 AREAS.

Compiler Action: The OPEN for the specified file is ignored.

Programmer Response: Probable user error. Delete the RESERVE clause from the file specification before recompiling. (Phase: GEN1)

CBL–0456 E NAME OR LITERAL MISSING BEFORE/insert/

Explanation: A keyword has been found where only an operand would be valid.

Compiler Action: Syntax checking continues as if a valid operand had been found, but either no code or arbitrary code is generated for the statement in error. Results are unpredictable.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN1, GENA, GEN2, GEN3)

CBL–0457 E INVALID TRANSACTION FILE TERMINAL ID LENGTH OR TYPE

Explanation: The terminal ID must be two characters in length; a data name specified as a terminal ID must be alphanumeric; a literal specified as a terminal ID must be nonnumeric.

Compiler Action: The entire statement is ignored.

Programmer Response: Probable user error. Correct the item before recompiling. (Phase: GEN1)

CBL–0458 E /insert/ IS NOT DEFINED AS A TRANSACTION FILE

Explanation: The file name specified on the ACQUIRE or DROP statement is not defined as a TRANSACTION file.

Compiler Action: The entire statement is ignored.

Programmer Response: Probable user error. Correct the invalid file specification before recompiling. (Phase: GEN1)

CBL–0459 E PRECEDING ELSE PHRASE INCOMPLETE--NEXT SENTENCE ASSUMED

Explanation: ELSE option does not contain a valid verb.

Compiler Action: Code for NEXT SENTENCE is generated.

Programmer Response: Correct the invalid ELSE option before recompiling. (Phase: GEN3)

CBL–0460 E MORE THAN 15 DELIMITERS SPECIFIED--/name or literal/IGNORED

Explanation: More than 15 delimiters are specified for an UNSTRING statement.

Compiler Action: All delimiters beyond the fifteenth are discarded without syntax check. Code is generated for the UNSTRING with the first 15 delimiters.

Programmer Response: Probable user error. Ensure that no more than 15 delimiters are specified for the UNSTRING statement before recompiling. (Phase: GEN2)

CBL–0461 E 'DELIMITER IN' IS NOT VALID UNLESS 'DELIMITED BY' IS SPECIFIED--/name or literal/IGNORED

Explanation: DELIMITER IN phrase is specified without the DELIMITED BY phrase for an UNSTRING statement.

Compiler Action: The DELIMITER IN phrase is discarded without syntax check. Code is generated for the UNSTRING statement without the DELIMITER IN phrase.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN2)

CBL–0462 E 'COUNT IN' IS NOT VALID UNLESS 'DELIMITED BY' IS SPECIFIED--/name or literal/IGNORED

Explanation: COUNT IN phrase is specified without the DELIMITED BY phrase for an UNSTRING statement.

Compiler Action: The COUNT IN phrase is discarded without syntax check. Code is generated for the UNSTRING statement without the COUNT IN phrase.

Programmer Response: Probable user error. Correct invalid statement before recompiling. (Phase: GEN2)

CBL–0463 C DELIMITER /name or literal/ EXCEEDS 1 CHARACTER--TRUNCATED

Explanation: For INSPECT REPLACING CHARACTERS, the size of a delimiter is limited to one character.

Compiler Action: The leftmost character of the delimiter is used.

Programmer Response: Reduce size of delimiter before recompiling. (Phase: GEN2)

CBL–0464 E REPLACING STRING /name or literal/ SMALLER THAN REPLACEMENT STRING

Explanation: In an INSPECT statement, the length of a string to replace exceeds the length of the replacing string.

Compiler Action: Code generation is discontinued but syntax checking continues for this statement.

Programmer Response: Probable user error. Correct size of strings so that they match. (Phase: GEN2)

CBL–0465 C REPLACING STRING /name or literal/ LARGER THAN REPLACEMENT STRING--TRUNCATED

Explanation: In an INSPECT statement, the length of a replacement string is exceeded by the length of the replacing string.

Compiler Action: The replacing string is truncated on the right to match the size of the replacement string.

Programmer Response: Probable user error. Correct size of strings so that they match. (Phase: GEN2)

CBL–0466 C TERMINAL ID INVALID--FOR CLAUSE IGNORED

Explanation: The terminal ID specified on an ACCEPT or DISPLAY statement must be a two-character alphanumeric field.

Compiler Action: The FOR clause is ignored.

Programmer Response: Probable user error. Correct the invalid terminal ID before recompiling. (Phase: GENA)

CBL–0467 C BOOLEAN COMPARISON MUST BE EQUAL OR NOT EQUAL--EQUAL ASSUMED

Explanation: The relation operator for a relation condition involving Boolean data must be equal or not equal.

Compiler Action: The test is generated with equal assumed.

Programmer Response: Probable user error. Correct the relation operator before recompiling. (Phase: GEN3)

CBL–0468 E INVALID FORMAT NAME SPECIFIED FOR A TRANSACTION FILE WRITE

Explanation: The FORMAT NAME must not be greater than eight characters; a data name specified as the FORMAT NAME must be alphanumeric; a literal specified as the FORMAT NAME must be nonnumeric.

Compiler Action: The entire statement is ignored.

Programmer Response: Probable user error. Correct the invalid item before recompiling. (Phase: GEN1)

CBL–0469 E INCORRECT LINE/LINES SPECIFIED FOR STARTING/ ROLLING/UP/DOWN CLAUSE

Explanation: The data name specified for lines must be defined as an elementary numeric item; the literal specified for lines must a numeric item.

Compiler Action: The entire statement is ignored.

Programmer Response: Probable user error. Correct invalid item before recompiling. (Phase: GEN1)

CBL–0470 E INVALID SYSTEM-NAME IN ASSIGN CLAUSE FOR S/34 COBOL--FILE IGNORED

Explanation: The system-name specified in the SELECT clause is a nonsupported device on System/34 COBOL.

Compiler Action: The file is ignored.

Programmer Response: Remove all references to the invalid device before recompiling. (Phase: DTA2)

CBL–0472 W /option/ OPTION NOT SUPPORTED FOR S/34 COBOL--OPTION IGNORED

Explanation: The option specified on this COBOL statement is not supported by System/34 COBOL.

Compiler Action: The option is ignored.

Programmer Response: Remove option before recompiling. (Phase: TEXT)

CBL–0473 W LIBRARY NAME INVALID OR LIBRARY NOT FOUND--LIBRARY IGNORED

Explanation: The library name specified with the OBJECT, LINK, or SUBLIB parameter of the PROCESS statement is invalid. The library name specified with the LIBRARY parameter on the PROCESS statement is invalid or cannot be located.

Compiler Action: The option specified on the PROCESS statement is ignored.

Programmer Response: Correct the invalid name specified before recompiling the job. (Phase: COPY)

CBL–0474 C KEYWORD /keyword/ INVALID IN THIS DIVISION--IGNORED

Explanation: A keyword has been found in the wrong division.

Compiler Action: The keyword is ignored.

Programmer Response: Remove keyword before recompiling. (Phase: TEXT)

CBL–0475 C EXTRANEOUS DATA ON THE USE STATEMENT--REST OF STATEMENT IGNORED

Explanation: Additional text was found on the USE statement.

Compiler Action: The additional text on the remainder of the USE statement is ignored.

Programmer Response: Remove the extraneous data before recompiling. (Phase: TEXT)

CBL–0476 C DECLARATIVES MISSING--STATEMENT IGNORED

Explanation: A declarative sentence (USE statement) was found without a preceding DECLARATIVE statement.

Compiler Action: The statement is ignored.

Programmer Response: Place the USE statement and procedure after a DECLARATIVES statement. (Phase: TEXT)

CBL–0477 C DECLARATIVES OR END DECLARATIVES HAS ALREADY BEEN PROCESSED--STATEMENT IGNORED

Explanation: Another DECLARATIVES statement has been detected.

Compiler Action: The DECLARATIVES statement is ignored.

Programmer Response: Remove th DECLARATIVES statement before recompiling. (Phase: TEXT)

CBL–0478 C NO DECLARATIVES SECTION--END DECLARATIVES IGNORED

Explanation: An END DECLARATIVES statement has been found without a matching DECLARATIVES statement.

Compiler Action: The END DECLARATIVES statement is ignored.

Programmer Response: Remove the END DECLARATIVES or supply the missing DECLARATIVES statement. (Phase: TEXT)

CBL–0479 E COPY NOT ALLOWED SINCE NOLIB INDICATED--STATEMENT IGNORED

Explanation: The NOLIB process option was specified which implies that COPY statements were not going to be used by this program.

Compiler action: The COPY statement is ignored.

Program Response: Remove the COPY statement from the program or remove the NOLIB option from the PROCESS option statement. (Phase: COPY, TEXT)

CBL–0480 E INVALID IDENTIFIER IN COPY STATEMENT--COPY IGNORED

Explanation: Invalid identifier found in COPY statement with REPLACING clause.

Compiler Action: The COPY statement is ignored.

Programmer Response: Correct the identifier used in the REPLACING clause of the COPY statement. (Phase: COPY)

CBL–0481 E INVALID WORD IN COPY STATEMENT--COPY IGNORED

Explanation: Invalid word found in COPY statement with REPLACING clause.

Compiler Action: The COPY statement is ignored.

Programmer Response: Correct the word used in the REPLACING clause of the COPY statement. (Phase: COPY)

CBL–0482 E INCOMPLETE IDENTIFIER OR PSEUDO-TEXT IN COPY STATEMENT--COPY IGNORED

Explanation: The identifier or pseudo-text in the REPLACING clause of a COPY statement was incomplete.

Compiler Action: The COPY statement is ignored.

Programmer Response: Correct the identifier or pseudo-text in the REPLACING clause of the COPY statement before recompiling. (Phase: COPY)

CBL–0483 E INCOMPLETE COPY STATEMENT--COPY IGNORED

Explanation: The COPY statement was not complete.

Compiler Action: The COPY statement is ignored.

Programmer Response: Complete the COPY statement before recompiling. (Phase: COPY)

CBL–0484 E OVERFLOW OF REPLACING TABLE--COPY IGNORED

Explanation: The REPLACING table was full; unable to process the REPLACING clause.

Compiler action: The COPY statement is ignored.

Programmer Response: Increase the region size for the compiler by using a REGION OCL statement before recompiling. (Phase: COPY)

CBL–0485 E EXPECTING 'BY' IN THE COPY STATEMENT--COPY IGNORED

Explanation: Keyword 'BY' was not found in the REPLACING clause of the COPY statement.

Compiler Action: The COPY statement is ignored.

Programmer Response: Add the keyword 'BY' to the REPLACING clause before recompiling. (Phase: COPY)

CBL–0486 E PSEUDO-TEXT-1 IS NULL IN THE COPY STATEMENT--COPY IGNORED

Explanation: Pseudo-text-1 in the REPLACING clause of the COPY statement was null. No replacement could be performed.

Compiler Action: The COPY statement is ignored.

Programmer Response: Correct the pseudo-text-1 in the REPLACING clause of the COPY statement before recompiling. (Phase: COPY)

CBL–0487 E AN ENDING DELIMITER IN THE COPY STATEMENT IS NOT FOLLOWED BY A VALID SEPARATOR--COPY IGNORED

Explanation: A valid separator does not follow the ending delimeter of pseudo-text in the REPLACING clause of the COPY statement.

Compiler Action: The COPY statement is ignored.

Programmer Response: Correct the REPLACING clause of the COPY statement by adding a valid separator before recompiling. (Phase: COPY)

CBL–0488 E INVALID ITEM IN AREA A OF THE COPY STATEMENT--COPY IGNORED

Explanation: Invalid clauses of the COPY statement were found starting in area A of the COBOL statement.

Compiler Action: The COPY statement is ignored.

Programmer Response: Correct the COPY statement by moving the clauses into area B before recompiling. (Phase: COPY)

CBL-0489 E INVALID USE OF COMMA OR SEMICOLON IN COPY STATEMENT--COPY IGNORED

Explanation: A comma or semicolon was found in an unexpected place in a COPY statement with REPLACING clause.

Compiler Action: The COPY statement is ignored.

Programmer Response: Remove the invalid comma or semicolon in the COPY statement before recompiling. (Phase: COPY)

CBL-0490 W DEBUGGING LINES INVALID WITHIN PSEUDO-TEXT-1--LINE IGNORED

Explanation: A debugging line was found in pseudo-text-1 of the REPLACING clause of the COPY statement.

Compiler Action: The debugging line is ignored and the COPY with REPLACING is performed.

Programmer Response: Remove the debugging line from pseudo-text-1 in the REPLACING clause of the COPY statement before recompiling. (Phase: COPY)

CBL-0491 W OVERFLOW IN COMMENT BUFFER

Explanation: The maximum of eight comment statements were exceeded in a text-name being copied into the COBOL program.

Compiler Action: Stop copying comment statements from text-name into the COBOL program.

Programmer Response: Remove some of the comments from text-name before recompiling. (Phase: COPY)

CBL-0495 C A QUOTE/APOSTROPHE MUST BE THE FIRST ELEMENT OF A NON-NUMERIC LITERAL

Explanation: A nonnumeric literal was preceded by a character other than a B for a Boolean literal.

Compiler Action: The literal is not processed by the compiler.

Programmer Response: Probable user error. Correct the character string so it is a valid COBOL literal before recompiling. (Phases: COPY, TEXT)

CBL-0496 E INVALID BOOLEAN LITERAL--LITERAL IGNORED

Explanation: A Boolean literal contained a numeric other than zero (0) or one (1).

Compiler Action: The Boolean literal is not processed by the compiler.

Programmer Response: Probable user error. Correct the character string so that it is a valid COBOL Boolean literal before recompiling. (Phase: TEXT)

CBL-0500 W WARNING-- /phrase/IS/level/ WHICH EXCEEDS THE FIPS LEVEL SPECIFIED

Explanation: The phrase is a higher level than specified on the LVL parameter of the PROCESS statement. The level of the indicated phrase is given by the level insert.

Compiler Action: Processing continues.

Programmer Response: If the program must be run with the level of support specified on the PROCESS statement, remove the indicated feature from the program.

CBL-0999 E FATAL COMPILER ERROR

Explanation: A disruptive error has occurred in the COBOL compiler.

Compiler Action: Compilation is discontinued.

Programmer Response: If this error continues to occur, contact your program support representative. (Phase: GEN1, GENA, GEN2, GEN3)

The following error messages are both printed on the compile listing and displayed on either the user's work station (if a work station is attached to the job) or the system operator's console (if a work station is not attached to the job). The formatted messages will appear with a message ID of CBL- preceding the messages. These messages all appear with only a 2-option available upon display, unless specifically noted as otherwise.

CBL–1000 COBOL COMPILER PHASE NOT FOUND

Explanation: The COBOL compiler was unable to find and load one of the COBOL compiler phases.

Operator Response: Install the COBOL Program Product again and rerun the job. If the problem persists, contact IBM for programming support.

CBL–1001 NO DATA OR PROCEDURE DIVISION FOUND

Explanation: The Procedure or Data Division was not found while compiling a COBOL program.

Operator Response: Select option 2.

CBL–1003 MORE THAN 65,535 STATEMENTS IN PROGRAM

Explanation: The COBOL compiler has detected that the program contains more than 65535 statements.

Operator Response: Select option 2.

CBL–1004 COPIED SOURCE MEMBER NOT FOUND IN LIBRARY

Explanation: The COBOL compiler attempted to copy a source member into the current source member being processed. However, the source member to be copied could not be found in the designated library.

Operator Response: Select option 2.

CBL–1005 MORE THAN 32,767 ENTRIES IN NAME TABLE

Explanation: The COBOL compiler has detected that the name table contains more than 32767 entries.

Operator Response: Select option 2.

CBL–1006 OBJECT PROGRAM EXCEEDS 65,535 BYTES

Explanation: The COBOL compiler has determined that the object program will exceed 65535 bytes.

Operator Response: Select option 2.

CBL–1007 PERMANENT ERROR READING PROGRAM SOURCE

Explanation: The COBOL compiler has detected a permanent I/O error while processing the input source member.

Operator Response: Select option 2.

CBL–1008 PERMANENT I/O ERROR ON PRINTER FILE

Explanation: The COBOL compiler has detected a permanent I/O error while printing the compiler output listing.

Operator Response: Select option 2.

CBL–1009 PERMANENT DISK I/O ERROR

Explanation: The COBOL compiler has detected a permanent I/O error on a COBOL compiler disk work file.

Operator Response: Select option 2.

CBL–1010 INSUFFICIENT STORAGE FOR COMPILATION

Explanation: There is insufficient storage to compile a COBOL program. A REGION OCL statement can be used to make more storage available.

Operator Response: Select option 2.

CBL–1011 INSUFFICIENT SPACE IN $WORK DISK FILE

Explanation: There is insufficient space in the $WORK disk file to compile your COBOL program.

Operator Response: Select option 2.

Programmer Response: Specify more blocks in the COMPILE OCL statement or in the COBOL, COBOLP, or COBOLCG procedure statements.

CBL–1012 INSUFFICIENT SPACE IN $SOURCE DISK FILE

Explanation: There is insufficient space in the $SOURCE disk file to compile your COBOL program.

Operator Response: Select option 2.

Programmer Response: Specify more blocks in the COMPILE OCL statement or in the COBOL, COBOLP, or COBOLCG procedure statements.

CBL–1013 INSUFFICIENT SPACE IN $WORK2 DISK FILE

Explanation: There is insufficient space in the $WORK2 disk file to compile your COBOL program.

Operator Response: Select option 2.

Programmer Response: Specify more blocks in the COMPILE OCL statement, or in the COBOL, COBOLP, or COBOLCG procedure statements.

CBL–1014 INSUFFICIENT SPACE IN $WORK3 DISK FILE

Explanation: There is insufficient space in the $WORK3 disk file to compile your COBOL program.

Operator Response: Select option 2.

Programmer Response: Specify more blocks in the COMPILE OCL statement, or in the COBOL, COBOLP, or COBOLCG procedure statements.

CBL–1015 INSUFFICIENT STORAGE FOR XREF PROCESSING

Explanation: Storage was insufficient to process a tag table or a reference table completely.

Compiler Action: The compilation has been terminated.

Programmer Response: Remove the XREF process option and recompile or specify a larger REGION size and recompile.

CBL–1016 SUBPROGRAM NAME TABLE EXCEEDS 20 NAMES

Explanation: The COBOL compiler has detected that the subprogram name table exceeds 20 subprogram names.

Operator Response: Select option 2.

CBL–1017 PATCH STACK EXCEEDED IN OBJ PHASE

Explanation: The number of unresolved patch references and definitions has exceeded the size of the patch stack. This condition can result from a lengthy, complex IF statement.

Compiler Action:

Option 2 - The job step is ended. Any new data created to this point is preserved and the job can continue with the next job step.

Programmer Response: If the condition is due to a lengthy, complex IF statement, simplifying this IF statement can eliminate this error. (Phase: OBJ)

CBL–1019 C OR E LEVEL DIAGNOSTICS DETECTED

Explanation: Error messages that are either C-level (conditional) or E-level (error) have been issued and the LET option has not been specified on the PROCESS statement.

Compiler Action:

Option 0 - Processing continues. The generated module is passed to the overlay linkage editor. However, the overlay linkage editor may not be able to link the program properly.

Option 2 - The job step is ended. Any new data created to this is preserved and the job can continue with the next job step.

Programmer Response: Correct all C-level and E-level errors.

CBL–1020 END OF COMPILATION

Explanation: This message is printed on the compiler output listing when the COBOL compiler has finished processing. This message is informational and does not require a response.

Programmer Response: No action is required.

CBL–1021 INSUFFICIENT STORAGE TO PROCESS ALL XREF NAMES

Explanation: Insufficient storage to enter all names into the sorted cross-reference listing.

Compiler Action:

Option 0 - Cross-reference processing is completed, but not all names will be listed, and the compilation continues.

Option 1 - Cross-reference processing is terminated with no listing, and the compilation continues.

Option 2 - The compilation is terminated, and message CBL-1015 is logged.

Option 3 - Entire job is terminated.

Programmer Response: Specify a larger REGION size and then recompile.

CBL–1022 INSUFFICIENT STORAGE TO PROCESS ALL XREF REFERENCES

Explanation: Insufficient storage to enter all references into the sorted cross-reference listing.

Compiler Action:

Option 0 - Cross-reference processing is completed but not all names will have a complete set of references. Compilation continues.

Option 1 - Cross-reference processing is terminated with no listing, and the compilation continues.

Option 2 - The compilation is terminated and message CBL-1015 is logged.

Option 3 - Entire job is terminated.

Programmer Response: Specify a larger REGION size and then recompile.

CBL–1099 UNKNOWN TERMINAL COMPILER ERROR

Explanation: This message is printed on the compiler output listing as shown. However, when the message is displayed, the message prefix is replaced with "ERRnn -", where nn is the error number.

Programmer Response: Select option 2, and record the value of nn for your program support representative.

COBOL DISPLAYED MESSAGES

COBOL messages that are displayed at a display station (COBOL object-time messages and displayed compiler messages) are not described in this manual. For information on COBOL displayed messages, see the *Displayed Messages Guide*.

Appendix B. Special Purpose Subroutines

Special purpose subroutines are provided to allow you to utilize special features of the System/34. These routines are as follows:

- CBMICR, CBMICO, CBEMCR, and CBEMCO
  - Read document information, using the 1255 MICR
- CBSTOP
  - Interrogates system shutdown status

1255 MAGNETIC INK CHARACTER READER (MICR) INTERFACE

The COBOL 1255 MICR interface subroutines provide the user with a method of accessing document information read by the 1255 MICR. The CBMICR and CBMICO subroutines provide a function equivalent to that found in SUBR08. The CBEMCR and CBEMCO subroutines provide a function equivalent to that found in SUBR25.

These subroutines provide two ways to process document information:

1. SUBR08
   System and stacker specifications describe the job to be done by the 1255.

2. SUBR25
   A device control language (DCL) program describes the job to be done by the 1255. The SUBR25 parameter list is the data management interface between SUBR25 and the DCL program. The parameter list takes the place of the system and stacker specifications in the COBOL program. The DCL program is a separate program that runs in the attachment I/O controller for the 1255.

The *1255 Magnetic Character Reader Reference Manual* contains a description of the following:

- SUBR08 and SUBR25
- System and stacker specifications
- SUBR25 Parameter List and Device Control Language program
- Input record format

The subroutines provide both an open and a read function. A call to an *open* subroutine (CBMICO or CBEMCO) is required before records can be read. When a call to a *read* subroutine (CBMICR or CBEMCR) returns an end-of-file condition, no more records can be read until a second *open* call has been executed. Formats of the subroutine calls are as follows:

- CALL 'CBEMCO' USING data-name-1
- CALL 'CBMICO' USING data-name-1
- CALL 'CBEMCR' USING data-name-2
- CALL 'CBMICR' USING data-name-2

Data-name-1 must refer to a structure having the following format:

```
01 Data-name-1.
   02 Data-name-a  PIC 9.
   02 Data-name-b  PIC 9(4)      USAGE IS COMP-4.
                                 VALUE IS integer-1.
   02 Data-name-c  PIC 9(4)      USAGE IS COMP-4.
                                 VALUE IS integer-2.
   02 Data-name-d  PIC 9(3)      USAGE IS COMP-4.
                                 VALUE IS integer-3.
   02 Data-name-e  PIC X (integer-2).
   02 Data-name-f.
   02 Data-name-g  PIC XX        VALUE IS
                                 HIGH-VALUE.
```

Data-name-a is the return code following each read operation. Return code values and meanings are:

0 – Successful completion

1 – End-of-file condition

3 – Permanent error

Data-name-b is the length of the system and stacker specifications or SUBR25 parameter list array (contained in Data-name-f). Integer-1 must be equal to or be a multiple of the number 80.

Data-name-c is the length of the input buffer (contained in Data-name-e). Integer-2 must be eight larger than the desired buffer size in bytes to allow for System/34 boundary alignment. The buffer must be at least large enough to accommodate ten records.

Data-name-d is the length of the input record. When this data structure is used with CBMICO, the length must be 55 bytes and is provided only for proper spacing of data.

Data-name-e is the input buffer. It must include eight additional positions for boundary alignment. The size must agree with the value coded in Data-name-c. No user references should be made to Data-name-e.

Data-name-f is the system and stacker specifications or SUBR25 parameter list array. The size of the array must agree with the value coded in Data-name-b. For a discussion of the array contents, refer to the *1255 Magnetic Character Reader Reference Manual*. Use the SUBR08 system and stacker specification format for CBMICO, and use the SUBR25 parameter list format for CBEMCO.

Data-name-g is the delimiter used by the *open* subroutine to check the accuracy of the structure. If your program contains a variable number of stacker specifications, be sure to move this field to the position following the last specification and place the proper value in Data-name-b before calling the *open* subroutine.

Data-name-2 is the logical record area into which the *read* subroutine places one input record each time the subroutine is called. The length must be at least 55 bytes for CBMICR and it must not be less than the value in Data-name-d for CBEMCR. Figure B-1 illustrates the format of the standard 55-character input record for CBMICR. The input record format for CBEMCR is user-defined.

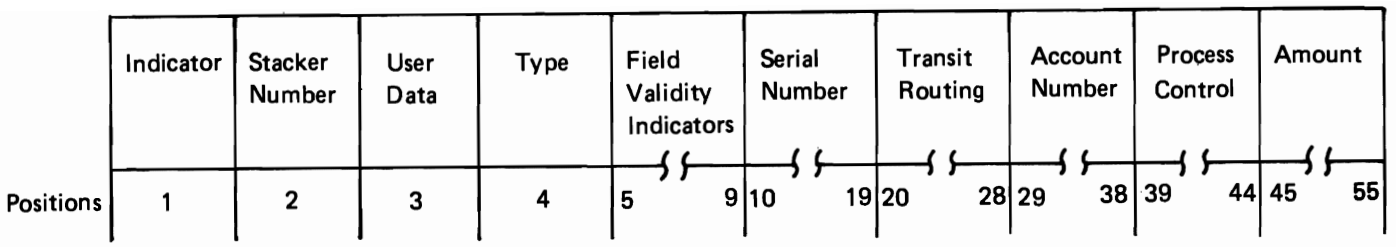


Figure B-1. Format of the Input Record

SHUTDOWN STATUS TEST

The CBSTOP subroutine is used to determine whether the system operator has requested system shutdown. This subroutine is called by the COBOL statement:

```
CALL 'CBSTOP' USING identifier
```

where identifier is a one-character numeric item defined in the Working-Storage or Linkage section of the calling program. Upon return from CBSTOP, the identifier will contain one of the following values:

0 – Shutdown has not been requested.

1 – Shutdown has been requested.

Appendix C. Language Summary and Comparison

ASSUMPTIONS FOR SYSTEM/34 COBOL LANGUAGE

1. The Low-Intermediate FIPS level of ANS 1974 COBOL is supported, with the exception of restrictions noted under *Indexed and Relative File Contents* in Chapter 8. This level requires the following processing modules: 1NUC, 1TBL, 1SEQ, 1REL, 1SEG, 1LIB, 1DEB, 1IPC.

   Summary of the Four Levels of FIPS COBOL

LOW	L/I	H/I	HIGH	
1	1	2	2	NUC–Nucleus
1	1	2	2	TBL–Table Handling
1	1	2	2	SEQ–Sequential I/O
-	1	2	2	REL–Relative I/O
-	-	-	2	INX–Indexed I/O
-	-	1	2	SRT–Sort-Merge
-	-	-	-	RPW–Report Writer
-	1	1	2	SEG–Segmentation
-	1	1	2	LIB–Library
-	1	2	2	DEB–Debug
-	1	2	2	IPC–Inter-Program Communication
1	-	2	2	COM–Communication

   Legend:

L/I	=	Low-Intermediate
H/I	=	High-Intermediate
-	=	Not included in the referenced level
1	=	The processing module must be implemented at ANS level 1
2	=	The processing module must be implemented at ANS level 2

2. A large selection of elements from higher level ANS modules are provided, as well as existing and new IBM extensions. System/3 COBOL and System/34 COBOL support requirements are used to complete this selection. Elements supported appear under the column System/34 in the *Summary of System/34 COBOL Language*.

3. As required by FIPS, a mechanism is provided for flagging elements that are not in a given FIPS level.

4. Compiler options are those provided by System/3 COBOL and System/34 PRPQ COBOL, as well as options for the following:

   - Cross-reference listing
   - Syntax-check only compile (no code generation)
   - Identification of statements that should not be included for FIPS level adherence

SUMMARY OF SYSTEM/34 COBOL LANGUAGE

The following description explains the headings and codes used in the summary of System/34 COBOL processing modules that appears on the following pages.

Column Headings

ANS 1	=	1974 ANS COBOL standard, level 1 of those modules considered for inclusion in System/34 COBOL
ANS 2	=	1974 ANS COBOL standard, level 2 of those modules considered for inclusion in System/34 COBOL
S/3	=	System/3 COBOL and System/34 PRPQ COBOL language (ANSI 1968 Standard)
S/34	=	System/34 COBOL language

Codes for Columns

X	Element is allowed (additional notes may apply)
-	Element is not allowed
a,b,c, ...	Indicated parenthetical note follows
1	Allowed, but treated as comments
2	Allowed, but with restrictions
3	System/3 compiler gives a diagnostic, but gives proper result
4	System/3 compiler supports the same function, but via different syntax
5	ANS 1974 standard indicates this is partly implementor-defined, or it is dependent on specific hardware components
6	Allowed, but IBM-defined limits exist (in accordance with note 5)
7	System/3 compiler diagnoses this as an error
8	System/3 compiler does not allow this to be omitted; if it is omitted, System/3 compiler gives a diagnostic but recovers according to 1974 ANS COBOL rules

Summary of Elements in the Nucleus

ANS 1	ANS 2	S/3	S/34	Elements
				LANGUAGE CONCEPTS
				Character Set
X	X	X	X	• Characters used for words: 0 through 9 and A through Z - (hyphen)
				• Characters used in punctuation:
X	X	X	X	Space ( ), equal sign (=), and quote (")
–	X	X	X	comma and semicolon
–	–	X	X	apostrophe instead of quote (**IBM Extension**)
				• Characters used in editing:
X	X	X	X	B + - . , Z * $
X	X	X	X	0 CR DB
X	X	–	X	/
–	X	X	X	• Characters used in arithmetic operations: + - * / **
–	X	X	X	• Characters used in relation conditions: = > <
				Separators
–	X	X	X	• Semicolon and comma
X	X	X	X	• Quote ("), period (.), and space ( ).
				Character-strings
				• COBOL words
X	X	X	X	• Words up to 30 characters are supported
				• User-defined words
X	X	X	X	• Data-name
–	X	X	X	• Data-name need not begin with an alpha character
X	X	X	X	• Level-number
X	X	X	X	• Mnemonic-name
X	X	X	X	• Paragraph-name
X	X	X	X	• Program-name
X	X	X	X	• Routine-name
X	X	X	X	• Section-name
–	X	X	X	• Condition-name
				• System-names
X	X	X	X	• Computer-name
X	X	X	X	• Implementor-name
X	X	X	X	• Language-name
				• Reserved words
X	X	X	X	• Key words
X	X	X	X	• Optional words

Summary of Elements in the Nucleus (Continued)

ANS 1	ANS 2	S/3	S/34	Elements
				• Figurative constants:
X	X	X	X	ZERO, SPACE
–	X	X	X	ZEROS, ZEROES, SPACES
X	X	X	X	HIGH-VALUE, LOW-VALUE, QUOTE
–	X	X	X	HIGH-VALUES, LOW-VALUES, QUOTES, ALL literal
–	X	X	X	• Special-character words: Arithmetic operators and relational operators
				• Connectives
–	X	X	X	• Qualifier connectives: OF, IN
–	X	X	X	• Series connectives: , (separator comma) ; (separator semicolon)
–	X	X	X	• Logical connectives: AND, OR, AND NOT, OR NOT
–	–	X	–	• Special register: TALLY (**ANS 1968**)
				• Literals
X	X	X	X	• Numeric literals: 1 to 18 digits
X	X	X	X	• Nonnumeric literals: 1 to 120 characters
X	X	X	X	• PICTURE character strings
X	X	X	X	• Comment-entries
				Qualification Rules
X	X	X	X	• Unqualified references to unique names
–	X	X	X	• Qualified references to nonunique names
–	X	X	X	• Data-names, paragraph-names, condition-names
–	X	–	X	• Text-names
				Reference Format
X	X	X	X	• Sequence number
				• Continuation of lines
X	X	X	X	• Nonnumeric literals
–	X	X	X	• Words and numeric literals
				• Comment lines
X	X	X	X	• Asterisk (*) comment line
X	X	X	X	• Stroke (/) comment line
				IDENTIFICATION DIVISION
X	X	X	X	• PROGRAM-ID paragraph
X	X	X	X	• AUTHOR paragraph
X	X	X	X	• INSTALLATION paragraph
X	X	X	X	• DATE-WRITTEN paragraph
–	X	–	X	• DATE-COMPILED paragraph
X	X	X	X	• SECURITY paragraph
–	–	X	–	• REMARKS paragraph (**ANS 1968**)

Summary of Elements in the Nucleus (Continued)

ANS 1	ANS 2	S/3	S/34	Elements
				ENVIRONMENT DIVISION
				Configuration Section
X	X	X	X	• SOURCE-COMPUTER paragraph
X	X	X	X	• OBJECT-COMPUTER paragraph.
X5	X5	X	X	• Computer-name
X5	X5	X	X	• MEMORY SIZE clause
X	X	–	X	• PROGRAM COLLATING SEQUENCE clause
X	X	X	X	• SPECIAL-NAMES paragraph
X	X	–	X	• Alphabet-name clause
X	X	–	X	• STANDARD-1 option
X	X	–	X	• NATIVE option
X5	X5	–	X	• Implementor-name option
–	X	–	X	• Literal option
X	X	X	X	• CURRENCY SIGN clause
X	X	X	X	• DECIMAL-POINT clause
X5	X5	X	X	• Implementor-name IS mnemonic-name
–	–	X	X	• IBM allows C01, CSP,
–	–	X	–	• IBM allows CONSOLE
–	–	–	X	• IBM allows REQUESTOR, SYSTEM-CONSOLE
–	–	X	X	• IBM allows UPSI-0 through UPSI-7
X	X	X	X	• ON STATUS IS condition-name
X	X	X	X	• OFF STATUS IS condition name
–	–	–	X	• IBM allows LOCAL-DATA, ATTRIBUTE-DATA, SYSTEM-SHUTDOWN
X	X	X	X	• Implementor-name series
				DATA DIVISION
				Working-Storage Section
X	X	X	X	• Data description entry
X	X	X	X	• BLANK WHEN ZERO clause
X	X	X	X	• Data-name or FILLER clause
X	X	X	X	• JUSTIFIED (Or JUST) clause
X	X	X	X	• Level-number
				• Valid (logical) values
X	X	X	X	• 01 through 10
–	X	X	X	• 11 through 49
–	X	–	X	• 66 (RENAMES)
X	X	X	X	• 77
–	X	X	X	• 88
				• Valid (physical) appearance
X	X	X	X	• Two digits supported (i.e. 01, 02, ...)
–	X	X	X	• One-digit abbreviation supported (i.e., 1, 2, ... 9)

Summary of Elements in the Nucleus (Continued)

ANS 1	ANS 2	S/3	S/34	Elements
X	X	X	X	• PICTURE (or PIC) clause
X	X	X	X	• Character-string may contain 30 characters
X	X	X	X	• Data characters: X 9
X	X	X	X	• Data character: A
X	X	X	X	• Operational symbol: S
X	X	X	X	• Operational symbol: V
X	X	X	X	• Operational symbol: P
				• Fixed insertion characters:
X	X	X	X	B + - . , $
X	X	X	X	0 CR DB
X	X	–	X	/
X	X	X	X	• Replacement or floating character $ + - Z *
X	X	X	X	• Currency sign substitution
X	X	X	X	• Decimal point substitution
X	X	X	X	• REDEFINES clause
–	X	X	X	• May be nested
–	X	–	X	• RENAMES clause
X	X	X	X	• SIGN clause
X5	X5	X1	X1	• SYNCHRONIZED (or SYNC) clause
X	X	X	X	• USAGE clause
X	X	X	X	• DISPLAY
X5	X5	X	X	• COMPUTATIONAL (or COMP)
–	–	X	X	• COMPUTATIONAL-3 (or COMP-3) (**IBM/Codaysl**)
–	–	X	X	• COMPUTATIONAL-4 (or COMP-4) (**IBM/Codaysl**)
X	X	X	X	• VALUE clause
X	X	X	X	• Literal
–	X	–	X	• Literal series
–	X	–	X	• Literal-1 THRU literal-2
–	X	–	X	• Literal range series
				PROCEDURE DIVISION
–	X5	Xa	Xa	Arithmetic Expressions (a=exponent identifier/literal must be positive integral value)
				Conditional Expressions
X	X	X	X	• Simple condition
X	X	X	X	• Relation condition
X	X	X	X	• Relational operators
X	X	X	X	• [NOT] GREATER THAN
–	X	X	X	• [NOT] >
X	X	X	X	• [NOT] LESS THAN
–	X	X	X	• [NOT] <
X	X	X	X	• [NOT] EQUAL TO
X	X	X	X	• [NOT] =

Summary of Elements in the Nucleus (Continued)

ANS 1	ANS 2	S/3	S/34	Elements
X	X	X	X	• Comparison of numeric operands
X	X	X	X	• Comparison of nonnumeric operands
–	X	X	X	• Operands of unequal size are permitted
X	X	X	X	• Class condition
–	X	X	X	• Condition-name condition
–	X	X	X	• Sign condition
X	X	X	X	• Switch-status condition
–	X	X	X	• Complex condition
–	X	X	X	• Logical Operators AND, OR, NOT
–	X	X	X	• Negated simple condition
–	X	X	X	• Combined and negated combined condition
–	X	–	X	• Abbreviated combined relation condition
X	X	X	X	Arithmetic Statements
X	X	X	X	• Arithmetic operands limited to 18 digits
–	X	–	X	• Multiple results in arithmetic statements
X	X	X	X	ACCEPT Statement
X5	X	X6	X6	• At least one transfer of data supported
–	X	X	X	• FROM mnemonic-name phrase
–	–	X	–	• FROM CONSOLE phrase (**IBM Extension**)
–	X	X	X	• FROM DATE phrase
–	X	–	X	• FROM DAY phrase
–	X	–	X	• FROM TIME phrase
X	X	X	X	ADD Statement
X	X	X	X	• Identifier/literal series
X	X	X	X	• TO identifier
–	X	–	X	• TO identifier series
X	X	X	X	• GIVING identifier
–	X	–	X	• GIVING identifier series
X	X	X	X	• ROUNDED phrase
X	X	X	X	• SIZE ERROR phrase
–	X	–	X	• CORRESPONDING phrase
X	X	X	X	ALTER Statement
X	X	X	X	• Procedure-name
–	X	X	X	• Procedure-name series
–	X	X	X	COMPUTE Statement
–	X5	X	X	• Arithmetic expression
–	X	–	X	• Identifier series
–	X	X	X	• ROUNDED phrase
–	X	X	X	• SIZE ERROR phrase

Summary of Elements in the Nucleus (Continued)

ANS 1	ANS 2	S/3	S/34	Elements
X	X	X	X	DISPLAY Statement
X5	X	X6	X6	• At least one transfer of data supported
–	X	–	X	• Multiple transfers of data supported
X	X	X	X	• Identifier/literal
X	X	X	X	• Identifier/literal series
–	X	X	X	• UPON mnemonic-name phrase
–	–	X	–	• UPON CONSOLE phrase (**IBM Extension**)
X	X	X	X	DIVIDE Statement
X	X	X	X	• INTO identifier
–	X	–	X	• INTO identifier series
X	X	X	X	• By identifier/literal
X	X	X	X	• GIVING identifier
–	X	–	X	• GIVING identifier series
–	X	–	X	• REMAINDER phrase
X	X	X	X	• ROUNDED phrase
X	X	X	X	• SIZE ERROR phrase
X	X	X	X	ENTER Statement
X	X	X	X	EXIT Statement
X	X	X	X	GO TO Statement
–	X	X	X	• Procedure-name may be omitted
X	X	X	X	• DEPENDING ON phrase
X	X	X	X	IF Statement
X	X	X	X	• Imperative-statement may contain multiple imperative verbs
–	X	X3	X	• Not limited to imperative statements
–	X	X3	X	• Nested statements
X	X	X	X	• ELSE phrase
X	X	X	X	• NEXT SENTENCE phrase
–	–	X	–	EXAMINE Statement (**ANS 1968**)
–	–	X	–	• Only single-character literals
–	–	X	–	• TALLYING phase
–	–	X	–	• ALL/LEADING/UNTIL FIRST option
–	–	X	–	• REPLACING phrase
–	–	X	–	• ALL/LEADING/FIRST/UNTIL FIRST option

Summary of Elements in the Nucleus (Continued)

ANS 1	ANS 2	S/3	S/34	Elements
X	X	–	X	INSPECT Statement
X	X	–	X	• Single-character identifiers or literals
–	X	–	X	• Multiple-character identifiers or literals
X	X	–	X	• TALLYING phrase
X	X	–	X	• BEFORE/AFTER INITIAL option
X	X	–	X	• REPLACING phrase
X	X	–	X	• TALLYING and REPLACING phrases
–	X	–	X	• TALLYING and REPLACING series
X	X	X	X	MOVE Statement
X	X	X	X	• TO identifier
X	X	X	X	• Identifier series
–	X	–	X	• CORRESPONDING phrase
X	X	X	X	MULTIPLY Statement
X	X	X	X	• BY identifier
–	X	–	X	• BY identifier series
X	X	X	X	• GIVING identifier
–	X	–	X	• GIVING identifier series
X	X	X	X	• ROUNDED phrase
X	X	X	X	• SIZE ERROR phrase
–	–	X	–	NOTE Statement (**ANS 1968**)
X	X	X	X	PERFORM Statement
X	X	X	X	• Procedure-name
X	X	X	X	• THRU phrase
X	X	X	X	• TIMES phrase
–	X	X	X	• UNTIL phrase
–	X	X2	X	• VARYING phrase
X	X	X	X	STOP Statement
X	X	X	X	• Literal
X	X	X	X	• RUN
–	X	–	X	STRING Statement
–	X	–	X	• Identifier/literal series
–	X	–	X	• DELIMITED BY phrase
–	X	–	X	• POINTER phrase
–	X	–	X	• ON OVERFLOW phrase

Summary of Elements in the Nucleus (Continued)

ANS 1	ANS 2	S/3	S/34	Elements
X	X	X	X	SUBTRACT Statement
X	X	X	X	• Identifier/literal series
X	X	X	X	• FROM identifier
–	X	–	X	• FROM identifier series
X	X	X	X	• GIVING identifier
–	X	–	X	• GIVING identifier series
X	X	X	X	• ROUNDED phrase
X	X	X	X	• SIZE ERROR phrase
–	X	–	X	• CORRESPONDING phrase
–	X	–	X	UNSTRING Statement
–	X	–	X	• DELIMITED BY phrase
–	X	–	X	• INTO series
–	X	–	X	• DELIMITER phrase
–	X	–	X	• COUNT phrase
–	X	–	X	• POINTER phrase
–	X	–	X	• TALLYING phrase
–	X	–	X	• ON OVERFLOW phrase

Summary of Elements in Table Handling Module

ANS 1	ANS 2	S/3	S/34	Elements
				LANGUAGE CONCEPTS
				User-Defined Words
X	X	X	X	• Index-name
				Subscripting
X	X	X	X	• 1 level supported
X	X	X	X	• 2 or 3 levels supported
				Indexing
X	X	X	X	• 1, 2, or 3 levels supported
				DATA DIVISION
X	X	X	X	OCCURS Clause
X	X	X	X	• Integer TIMES
–	X	–	X	• Integer-1 TO integer-2 DEPENDING on data-name
–	X	–	X	• ASCENDING/DESCENDING data-name
–	X	–	X	• Data-name series
–	X	–	X	• ASCENDING/DESCENDING series
X	X	X	X	• INDEXED BY index-name series
X5	X5	X	X	USAGE IS INDEX Clause
				PROCEDURE DIVISION
–	X	–	X	SEARCH Statement
–	X	–	X	• VARYING phrase
–	X	–	X	• AT END phrase
–	X	–	X	• WHEN phrase
–	X	–	X	• WHEN phrase series
–	X	–	X	• ALL phrase
–	X	–	X	• WHEN phrase
–	X	–	X	• AT END phrase
X	X	X	X	SET Statement
X	X	X	X	• Index-name/identifier series
X	X	X	X	• Index-name
X	X	X	X	• UP BY identifier/integer
X	X	X	X	• DOWN BY identifier/integer
X	X	X	X	• Index-name series

Summary of Elements in the Sequential I-O Module

ANS 1	ANS 2	S/3	S/34	Elements
				LANGUAGE CONCEPTS
				User-Defined Words
X	X	X	X	• File-name
X	X	X	X	• Record-name
X	X	–	X	I-O Status
				Special Register
–	X	X2	X	• LINAGE-COUNTER
				ENVIRONMENT DIVISION
				INPUT-OUTPUT SECTION
X	X	X	X	• FILE-CONTROL paragraph
X	X	X	X	• File control entry
X	X	X	X	• SELECT clause
–	X	–	X	• OPTIONAL phrase
X	X	X	X	• ASSIGN clause
–	–	X	–	• FOR MULTIPLE REEL/UNIT phrase (**ANS 1968**)
X	X	X4	X	• ORGANIZATION IS SEQUENTIAL clause
X	X	X	X	• ACCESS MODE IS SEQUENTIAL clause
X	X	–	X	• FILE STATUS clause
–	X	–	X	• RESERVE integer AREA(S) clause
–	–	X	–	• RESERVE NO/integer ALTERNATE AREA(S) clause (** ANS 1968**)
–	–	X1	–	• PROCESSING MODE clause (**ANS 1968**)
–	–	X1	–	• FILE-LIMIT clause (**ANS 1968**)
X	X	X	X	• I-O-CONTROL paragraph
X	X	X2	X	• RERUN clause
X	X	X	X	• SAME AREA clause
X	X	X	X	• SAME AREA series
–	X	–	X	• SAME RECORD AREA clause
–	X	–	X	• SAME RECORD AREA series
–	X5	–	X1	• MULTIPLE FILE TAPE clause

Summary of Elements in the Sequential I-O Module (Continued)

ANS 1	ANS 2	S/3	S/34	Elements
				DATA DIVISION
				FILE SECTION
X	X	X	X	• File description entry
X	X	X	X	• Record description entry
X	X	X	X	• BLOCK CONTAINS clause
X	X	X	X	• Integer RECORDS/CHARACTERS
–	X	–	X	• Integer-1 TO integer-2 RECORDS/CHARACTERS
X5	X5	–	X1	• CODE-SET clause
X	X	X1	X1	• DATA RECORDS clause
X	X	X1	X1	• Data-name
X	X	X1	X1	• Data-name series
X	X	X	X	• LABEL RECORDS clause
X	X	X	X	• STANDARD
X	X	X7	X	• OMITTED
–	X	X2	X	• LINAGE clause
–	X	X2	X	• FOOTING phrase
–	X	–	X	• TOP phrase
–	X	–	X	• BOTTOM phrase
X	X	X	X	• RECORD CONTAINS clause
X	X	X	X	• Integer-1 TO integer-2 CHARACTERS
X5	X5	X1	X1	• VALUE OF clause
X5	X5	X1	X1	• Implementor-name IS literal
X	X	–	X1	• Implementor-name IS literal series
–	X	–	X1	• Implementor-name IS data-name
–	X	–	X1	• Implementor-name IS data-name series
				PROCEDURE DIVISION
X	X	X	X	CLOSE Statement
X	X	X	X	• Single file-name
–	X	X	X	• File-name series
X5	X5	X	X1	• REEL/UNIT
–	X5	X	X	• WITH LOCK phrase
–	X5	–	X1	• WITH NO REWIND phrase
–	X5	–	X1	• FOR REMOVAL phrase
X	X	X	X	OPEN Statement
X	X	X	X	• Single file-name
–	X	X	X	• File-name series
X	X	X	X	• INPUT phrase
–	X5	–	X1	• REVERSED phrase
–	X5	–	X1	• WITH NO REWIND phrase

Summary of Elements in Sequential I-O Module (Continued)

ANS 1	ANS 2	S/3	S/34	Elements
X	X	X	X	• OUTPUT phrase
–	X5	–	X1	• NO REWIND phrase
X5	X5	X	X	• I-O phrase
–	X5	–	X	• EXTEND phrase
–	X	–	X	• INPUT, OUTPUT, I-O, EXTEND series
X	X	X	X	READ Statement
X	X	X	X	• INTO identifier
X	X	X8	X	• AT END phrase
X5	X5	X4	X	REWRITE Statement
X	X	X	X	• FROM identifier
X	X	–	X	USE Statement
X	X	–	X	• EXCEPTION/ERROR PROCEDURE phrase
X	X	–	X	• ON file-name
X	X	–	X	• ON INPUT
X	X	–	X	• ON OUTPUT
X	X	–	X	• ON I-O
–	X	–	X	• ON EXTEND
–	X	–	X	• ON file-name series
X	X	X	X	WRITE Statement
X	X	X	X	• FROM identifier
X	X5	X	X	• BEFORE/AFTER ADVANCING phrase integer
X5	X5	X	X	• Integer
–	X5	X	X	• Identifier
X5	X5	X	X	• LINE(S) option
X5	X5	X	X	• PAGE
–	X5	X	X	• Mnemonic-name
–	X5	X	X	• AT END-OF-PAGE/EOP phrase
–	–	X	–	• INVALID KEY phrase (**ANS 1968**)

Summary of Elements in the Relative I-O Module

ANS 1	ANS 2	S/3	S/34	Elements
				LANGUAGE CONCEPTS
				User-Defined Words
X	X	X	X	• File-name
X	X	X	X	• Record-name
X	X	–	X	I-O Status
				ENVIRONMENT DIVISION
				INPUT-OUTPUT SECTION
X	X	X	X	• FILE-CONTROL paragraph
X	X	X	X	• File control entry
X	X	X	X	• SELECT clause
X	X	X	X	• ASSIGN clause
X	X	X4	X	• ORGANIZATION IS RELATIVE clause
X	X	X	X	• ACCESS MODE clause
X	X	X	X	• SEQUENTIAL
X	X	–	X	• RELATIVE KEY phrase
X	X	X	X	• RANDOM
X	X	–	X	• RELATIVE KEY phrase
–	–	X	–	• ACTUAL KEY phrase (**ANS 1968**)
–	X	–	X	• DYNAMIC
–	X	–	X	• RELATIVE KEY phrase
X	X	–	X	• FILE STATUS clause
–	X	–	X	• RESERVE integer AREA(S) clause
–	–	X	–	• RESERVE NO/integer ALTERNATE AREA(S) clause (** ANS 1968**)
–	–	X1	–	• FILE-LIMIT clause (**ANS 1968**)
–	–	X1	–	• PROCESSING MODE clause (**ANS 1968**)
X	X	X	X	• I-O-CONTROL paragraph
X	X	X	X	• RERUN clause
X	X	X	X	• SAME AREA clause
X	X	X	X	• SAME AREA series
–	X	–	X	• SAME RECORD AREA clause
–	X	–	X	• SAME RECORD AREA series

Summary of Elements in Relative I-O Module (Continued)

ANS 1	ANS 2	S/3	S/34	Elements
				DATA DIVISION
				FILE SECTION
X	X	X	X	• File description entry
X	X	X	X	• Record description entry
X	X	X	X	• BLOCK CONTAINS clause
X	X	X	X	• Integer RECORDS/CHARACTERS
–	X	–	X	• Integer-1 TO integer-2 RECORDS/CHARACTERS
X	X	X1	X1	• DATA RECORDS clause
X	X	X1	X1	• Data-name
X	X	X1	X1	• Data-name series
X	X	X	X	• LABEL RECORDS clause
X	X	X	X	• STANDARD
X	X	X7	X	• OMITTED
X	X	X	X	• RECORD CONTAINS clause
X	X	X	X	• Integer-1 TO integer-2 CHARACTERS
X5	X5	X1	X1	• VALUE OF clause
X5	X5	X1	X1	• Implementor-name IS literal
X5	X5	–	X1	• Implementor-name IS literal series
–	X5	–	X1	• Implementor-name IS data-name
–	X5	–	X1	• Implementor-name IS data-name series
				PROCEDURE DIVISION
X	X	X	X	CLOSE Statement
X	X	X	X	• Single file-name
X	X	X	X	• File-name series
X5	X5	X	X	• WITH LOCK phrase
X	X	–	X	DELETE Statement
X	X	–	X	• INVALID KEY phrase
X	X	X	X	OPEN Statement
X	X	X	X	• Single file-name
X	X	X	X	• File-name series
X	X	X	X	• INPUT phrase
X	X	X	X	• OUTPUT phrase
X5	X5	X	X	• I-O phrase
X	X	X	X	• INPUT, OUTPUT, and I-O series

Summary of Elements in Relative I-O Module (Continued)

ANS 1	ANS 2	S/3	S/34	Elements
X	X	X	X	READ Statement
X	X	X	X	• INTO identifier
–	X	–	X	• NEXT phrase
X	X	X8	X	• AT END phrase
X	X	X8	X	• INVALID KEY phrase
X5	X5	X	X	REWRITE Statement
X	X	X	X	• FROM identifier
X	X	X	X	• INVALID KEY phrase
–	–	X1	–	SEEK Statement (**ANS 1968**)
–	X	–	X	START Statement
–	X	–	X	• KEY IS phrase
–	X	–	X	• INVALID KEY phrase
X	X	–	X	USE Statement
X	X	–	X	• EXCEPTION/ERROR PROCEDURE phrase
X	X	–	X	• ON file-name
X	X	–	X	• ON INPUT
X	X	–	X	• ON OUTPUT
X	X	–	X	• ON I-O
–	X	–	X	• ON file-name series
X	X	X	X	WRITE Statement
X	X	X	X	• FROM identifier
X	X	X8	X	• INVALID KEY phrase

Summary of Elements in Indexed I-O Module

ANS 1	ANS 2	S/3	S/34	Elements
				LANGUAGE CONCEPTS
				User-Defined Words
X	X	X	X	• File-name
X	X	X	X	• Record-name
X	X	–	X	I-O Status
				ENVIRONMENT DIVISION
				INPUT-OUTPUT SECTION
X	X	X	X	• FILE-CONTROL paragraph
X	X	X	X	• File control entry
X	X	X	X	• SELECT clause
X	X	X	X	• ASSIGN clause
X	X	X4	X	• ORGANIZATION IS INDEXED clause
X	X	X	X	• ACCESS MODE clause
X	X	X	X	• SEQUENTIAL
X	X	X	X	• RANDOM
–	X	–	X	• DYNAMIC
X	X	X	X	• RECORD KEY clause
–	X	–	–	• ALTERNATE RECORD KEY clause
–	X	–	–	• WITH DUPLICATES phrase
–	–	X	–	• NOMINAL KEY clause (**IBM Extension**)
X	X	–	X	• FILE STATUS clause
–	X	X	X	• RESERVE integer AREA(S) clause
–	–	X	–	• RESERVE NO/integer ALTERNATE AREA(S) clause (** ANS 1968**)
–	–	X1	–	• FILE-LIMIT clause (**ANS 1968**)
–	–	X1	–	• PROCESSING MODE IS clause (**ANS 1968**)
X	X	X	X	• I-O-CONTROL paragraph
X	X	X	X	• RERUN clause
X	X	X	X	• SAME AREA clause
X	X	X	X	• SAME AREA series
–	X	–	X	• SAME RECORD AREA clause
–	X	–	X	• SAME RECORD AREA series
–	–	X	X	• APPLY CORE-INDEX series (**IBM Extension**)

Summary of Elements in Indexed I-O Module (Continued)

ANS 1	ANS 2	S/3	S/34	Elements
				DATA DIVISION
				FILE SECTION
X	X	X	X	• File description entry
X	X	X	X	• Record description entry
X	X	X	X	• BLOCK CONTAINS clause
X	X	X	X	• Integer RECORDS/CHARACTERS
–	X	–	X	• Integer-1 TO integer-2 RECORDS/CHARACTERS
X	X	X1	X1	• DATA RECORDS clause
X	X	X1	X1	• Data-name
X	X	X1	X1	• Data-name series
X	X	X	X	• LABEL RECORDS clause
X	X	X	X	• STANDARD
X	X	X7	X	• OMITTED
X	X	X	X	• RECORD CONTAINS clause
X	X	X	X	• Integer-1 TO integer-2 CHARACTERS
X5	X5	X1	X1	• VALUE OF clause
X5	X5	X1	X1	• Implementor-name IS literal
X5	X5	–	X1	• Implementor-name IS literal series
–	X5	–	X1	• Implementor-name IS data-name
–	X5	–	X1	• Implementor-name IS data-name series
				PROCEDURE DIVISION
X	X	X	X	CLOSE Statement
X	X	X	X	• Single file-name
X	X	X	X	• File-name series
X5	X5	X	X	• WITH LOCK phrase
X	X	–	X	DELETE Statement
X	X	–	X	• INVALID KEY phrase
X	X	X	X	OPEN Statement
X	X	X	X	• Single file-name
X	X	X	X	• File-name series
X	X	X	X	• INPUT phrase
X	X	X	X	• OUTPUT phrase
X5	X5	X	X	• I-O phrase
X	X	X	X	• INPUT, OUTPUT, and I-O series

Summary of Elements in Indexed I-O Module (Continued)

ANS 1	ANS 2	S/3	S/34	Elements
X	X	X	X	READ Statement
X	X	X	X	• INTO identifier
–	X	–	X	• KEY IS phrase
–	X	–	X	• NEXT phrase
X	X	X8	X	• AT END phrase
X	X	X8	X	• INVALID KEY phrase
X5	X5	X	X	REWRITE Statement
X	X	X	X	• FROM identifier
X	X	X8	X	• INVALID KEY phrase
–	X	X	X	START Statement
–	X	X4	X	• KEY IS phrase
–	X	X8	X	• INVALID KEY phrase
X	X	–	X	USE Statement
X	X	–	X	• EXCEPTION/ERROR PROCEDURE phrase
X	X	–	X	• ON file-name
X	X	–	X	• ON INPUT
X	X	–	X	• ON OUTPUT
X	X	–	X	• ON I-O
–	X	–	X	• ON file-name series
X	X	X	X	WRITE Statement
X	X	X	X	• FROM identifier
X	X	X8	X	• INVALID KEY phrase

Summary of Elements in the Sort-Merge Module

ANS 1	ANS 2	S/3	S/34	Elements
				LANGUAGE CONCEPTS
				User-Defined Words
X	X	–	X	• File-name
				ENVIRONMENT DIVISION
				INPUT-OUTPUT SECTION
X	X	–	X	• FILE-CONTROL paragraph
X	X	–	X	• File control entry
X	X	–	X	• SELECT clause
X	X	–	X	• ASSIGN clause
–	X	–	X	• I-O-CONTROL paragraph
–	X	–	X	• SAME RECORD AREA clause
–	X	–	X	• SAME RECORD AREA series
–	X	–	X	• SAME SORT/SORT-MERGE AREA clause
–	X	–	X	• SAME SORT/SORT-MERGE AREA series
				DATA DIVISION
				FILE SECTION
X	X	–	X	• File description entry
X	X	–	X	• Record description entry
X	X	–	X	• DATA RECORDS clause
X	X	–	X	• Data-name
X	X	–	X	• Data-name series
X	X	–	X	• RECORD CONTAINS clause
X	X	–	X	• Integer-1 TO integer-2 CHARACTERS
				PROCEDURE DIVISION
–	X	–	X	MERGE Statement
–	X	–	X	• KEY data-name
–	X	–	X	• Data-name series
–	X	–	X	• ASCENDING series
–	X	–	X	• DESCENDING series
–	X	–	X	• Mixed ASCENDING/DESCENDING
–	X	–	X	• COLLATING SEQUENCE phrase
–	X	–	X	• USING phrase
–	X	–	X	• OUTPUT PROCEDURE phrase
–	X	–	X	• GIVING phrase

Summary of Elements in the Sort-Merge Module (Continued)

ANS 1	ANS 2	S/3	S/34	Elements
X	X	–	X	RELEASE Statement
X	X	–	X	• FROM phrase
X	X	–	X	RETURN Statement
X	X	–	X	• INTO phrase
X	X	–	X	• AT END phrase
X	X	–	X	SORT Statement
X	X	–	X	• Program may contain one SORT statements
–	X	–	X	• Program may contain multiple SORT statement
X	X	–	X	• KEY data-name
X	X	–	X	• Data-name series
X	X	–	X	• ASCENDING series
X	X	–	X	• DESCENDING series
X	X	–	X	• Mixed ASCENDING/DESCENDING
–	X	–	X	• COLLATING SEQUENCE phrase
X	X	–	X	• INPUT PROCEDURE phrase
X	X	–	X	• USING phrase
X	X	–	X	• OUTPUT PROCEDURE phrase
X	X	–	X	• GIVING phrase

Summary of Elements in the Debug Module

ANS 1	ANS 2	S/3	S/34	Elements
				LANGUAGE CONCEPTS
				Special Registers
X	X	–	X	• DEBUG-ITEM
				ENVIRONMENT DIVISION
				CONFIGURATION SECTION
				• SOURCE-COMPUTER paragraph
X	X	–	X	• WITH DEBUGGING MODE clause
				PROCEDURE DIVISION
X	X	–	X	USE FOR DEBUGGING Statement
X	X	–	X	• Procedure-name
X	X	–	X	• Procedure-name series
X	X	–	X	• ALL PROCEDURES
–	X	–	–	• ALL REFERENCES OF identifier series
–	X	–	–	• File-name series
–	X	–	–	• Cd-name series
–	–	X	X	EXHIBIT Statement (**IBM Extension**)
–	–	X	X	• NAMED/CHANGED NAMED option
–	–	X	X	• Identifier series
–	–	X	X	READY TRACE Statement (**IBM Extension**)
–	–	X	X	RESET TRACE Statement (**IBM Extension**)
				ALL DIVISIONS
X	X	–	X	Debugging Lines (permitted after the OBJECT-COMPUTER paragraph)

Summary of Elements in the Inter-Program Communication Module

ANS 1	ANS 2	S/3	S/34	Elements
				DATA DIVISION
X	X	X	X	Linkage Section
				PROCEDURE DIVISION
				Procedure Division Header
X	X	X	X	• USING phrase (a=At least five data-names must be supported)
X	X	X	X	CALL Statement
X	X	X	X	• Literal
–	X	–	–	• Identifier
X	X	X	X	• USING phrase (a=At least five data-names must be supported)
–	X	–	–	• ON OVERFLOW phrase
–	X	–	–	CANCEL Statement
X	X	X	X	EXIT PROGRAM Statement

Summary of Elements in the Segmentation Module

ANS 1	ANS 2	S/3	S/34	Elements
				LANGUAGE CONCEPTS
				User-Defined Words
X	X	X	X	• Segment-number
				ENVIRONMENT DIVISION
				CONFIGURATION SECTION
				• OBJECT-COMPUTER paragraph
–	X	–	X	• SEGMENT-LIMIT clause
				PROCEDURE DIVISION
X	X	X	X	Segment-Numbers
X	X	X	X	• Fixed segment-number 0-49
X	X	X	X	• Independent segment-number 50-99
–	X	–	X	• Sections with same number need not be contiguous

Summary of Elements in the Library Module

ANS 1	ANS 2	S/3	S/34	Elements
				LANGUAGE CONCEPTS
				User-Defined Words
X5	X5	X	X	• Text-name
–	X5	–	X	• Library-name
				ALL DIVISIONS
X	X	X	X	COPY Statement
X	X	X	X	• Text-name
–	X	–	X	• OF/IN library-name
–	X	–	X	• REPLACING phrase

Appendix D: Intermediate Result Fields

This appendix discusses the conceptual compiler algorithms for determining the number of integer and decimal places reserved for intermediate results. The following abbreviations are used:

i – number of integer places carried for an intermediate result.

d – number of decimal places carried for an intermediate result.

dmax – in a particular statement, the larger of either:

- The number of decimal places needed for the final result field.
- The maximum number of decimal places defined for any operand except exponents and divisors.

op1 – first operand in a generated arithmetic statement.

op2 – second operand in a generated arithmetic statement.

d1,d2 – number of decimal places defined for op1 or op2, respectively.

ir – intermediate result field obtained from the execution of a generated arithmetic statement or operation. Ir1, ir2, and so on represent successive intermediate results. These intermediate results are generated either in registers or in storage locations. Successive intermediate results can have the same location.

When an arithmetic statement contains only a single pair of operands, no intermediate results are generated. Intermediate results are possible in the following cases:

1. In an ADD or SUBTRACT statement containing multiple operands immediately following the verb
2. In a COMPUTE statement specifying a series of arithmetic operations
3. In arithmetic expressions contained in an IF or PERFORM statement
4. In the GIVING option with multiple result fields for the ADD, SUBTRACT, MULTIPLY, DIVIDE, or COMPUTE statements

In such cases, the compiler treats the statement as a succession of operations. For example, the following statement:

```
COMPUTE Y = A + B * C - D / E + F ** G
```

is replaced by

```
F**G                              yielding ir1
MULTIPLY B      BY C              yielding ir2
DIVIDE E        INTO D            yielding ir3
ADD A           TO ir2            yielding ir4
SUBTRACT ir3    FROM ir4          yielding ir5
ADD ir5         TO ir1            yielding Y
```

Compiler Calculation of Intermediate Results

The number of integer places in an intermediate result (ir) is calculated as follows:

- The compiler first determines the maximum value that the ir can contain by performing the statement in which the ir occurs.
  - If an operand in this statement is a data-name, the value used for the data-name is equal to the numerical value of the PICTURE for the data-name (for example, PICTURE 9V99 has the value 9.99).
  - If an operand is a literal, the actual value of the literal is used.
  - If an operand is an intermediate result, the value determined for the intermediate result in a previous arithmetic operation is used.
  - If the operation is division:
    a. If op2 is a data-name, the value used for op2 is the minimum nonzero value of the digit in the PICTURE for the data-name (for example, PICTURE 9V99 has the value 0.01).
    b. If op2 is an intermediate result, the intermediate result is treated as though it had a PICTURE, and the minimum nonzero value of the digits in this PICTURE is used.
    c. If the ROUNDED option is used, the number of decimal places in an ir increases by 1.

- When the maximum value of the ir is determined by the above procedures, i is set equal to the number of integers in the maximum value.

- The number of decimal places contained in an ir is calculated as:

Operation	Decimal Places
+ or -	d1 or d2, whichever is greater
*	d1 + d2
/	d1 -.d2 or dmax, whichever is greater
**	dmax if op2 is nonintegral or a data-name; d1 * op2 if op2 is an integral literal

Note: The user must define the operands of any arithmetic statement with enough decimal places to give the desired accuracy in the final result.

The following illustration indicates the action of the compiler when handling intermediate results:

Value of i + d	Value of d	Value of i + dmax	Action Taken
<19 = 19	Any Value	Any value	*i* integer and *d* decimal places are carried for ir
>19	<dmax	Any value	19 - *d* integer and *d* decimal places are carried for ir (See *Note.*)
	=dmax		
	>dmax	<19	*i* integer and 19 - *i* decimal places are carried for ir
		= 19	
		>19	19 - dmax integer and dmax decimal places are carried for ir (See *Note.*)
Note: High-order integers may be truncated.			

Figure D-1. Compiler Action on Intermediate Results

Appendix E. Sample File-Processing Programs

The programs in this appendix illustrate the fundamental programming techniques associated with each type of file organization. They are intended to be used for planning purposes only, and to illustrate the input/output statements necessary for certain access methods. Other COBOL features (the use of condition-names and the PERFORM statement, for example) are used only incidentally. The programs are:

- Sequential File Creation
- Sequential File Updating and Extension
- Indexed File Creation
- Indexed File Updating
- Relative File Creation
- Relative File Updating
- Relative File Retrieval
- COBOL SORT Example

Sequential File Creation

This program creates a sequential file by extracting employee salary records. The input records are arranged in ascending order by employee number. The output file has the identical order.

Note: The program-id used by the system is CRTSEQ.

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S .........IDENTFCN SEQ/NO S

     PROCESS SOURCE                                                          0001
   1 IDENTIFICATION DIVISION.                                                0002
   2 PROGRAM-ID.  CRTSEQUENTIAL.                                             0003
                                                                             0004
   3 ENVIRONMENT DIVISION.                                                   0005
   4 CONFIGURATION SECTION.                                                  0006
   5 SOURCE-COMPUTER.  IBM-S34.                                              0007
   6 OBJECT-COMPUTER.  IBM-S34.                                              0008
   7 INPUT-OUTPUT SECTION.                                                   0009
   8 FILE-CONTROL.                                                           0010
   9     SELECT OUT-FILE                                                     0011
             ASSIGN TO DISK-MSTFILE.                                         0012
                                                                             0013
  10 DATA DIVISION.                                                          0014
  11 FILE SECTION.                                                           0015
  12 FD  OUT-FILE                                                            0016
         LABEL RECORDS ARE STANDARD.                                         0017
  13 01  OUT-REC.                                                           0018
  14     05  OUT-RECTYPE              PICTURE XX.                           0019
  15     05  OUT-EMPLOYEE-NUMBER      PICTURE 9(6).                         0020
  16     05  OUT-EMPLOYEE-NAME        PICTURE X(28).                        0021
  17     05  OUT-EMPLOYEE-CODE        PICTURE 9.                            0022
  18     05  OUT-EMPLOYEE-SALARY      PICTURE 9(6)V99.                      0023
  19 WORKING-STORAGE SECTION.                                                0024
  20 01  INP-REC.                                                           0025
  21     05  INP-RECTYPE              PICTURE XX.                           0026
  22         88  INP-EMPLOYEE         VALUES "H1" THRU "H5".                0027
  23         88  THE-END-OF-INPUT     VALUE "/*".                           0028
  24     05  INP-EMPLOYEE-NUMBER      PICTURE 9(6).                         0029
  25     05  INP-EMPLOYEE-NAME        PICTURE X(28).                        0030
  26     05  INP-EMPLOYEE-CODE        PICTURE 9.                            0031
  27     05  INP-EMPLOYEE-SALARY      PICTURE 9(6)V99.                      0032
                                                                             0033
  28 PROCEDURE DIVISION.                                                     0034
                                                                             0035
  29 TOP-LOGIC-PARA.                                                         0036
  30     OPEN OUTPUT OUT-FILE.                                               0037
  31     ACCEPT INP-REC.                                                     0038
  32     PERFORM PROCESS-DATA UNTIL THE-END-OF-INPUT.                        0039
  33     CLOSE OUT-FILE.                                                     0040
  34     STOP RUN.                                                           0041
                                                                             0042
  35 PROCESS-DATA.                                                           0043
  36     IF INP-EMPLOYEE                                                     0044
  37         WRITE OUT-REC FROM INP-REC                                      0045
  38         ELSE DISPLAY " INVALID RECORD --->" INP-REC.                    0046
  40     ACCEPT INP-REC.                                                     0047
```

Sequential File Updating and Extension

This program updates and extends the MST-FILE. The INP-FILE, which was also created by CRTSEQUENTIAL, and the MST-FILE area each read. When a match is found between INP-EMPLOYEE-NUMBER and MST-EMPLOYEE-NUMBER, the input record replaces the original record. After the MST-FILE has been completely processed, new employee records are added at the end of the file.

Note: The program-id used by the system is UPDTSE.

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S .........IDENTFCN SEQ/NO S

     PROCESS SOURCE                                                                0001
   1 IDENTIFICATION DIVISION.                                                      0002
   2 PROGRAM-ID.  UPDTSEQUENTIAL.                                                  0003
                                                                                   0004
   3 ENVIRONMENT DIVISION.                                                         0005
   4 CONFIGURATION SECTION.                                                        0006
   5 SOURCE-COMPUTER.  IBM-S34.                                                    0007
   6 OBJECT-COMPUTER.  IBM-S34.                                                    0008
   7 INPUT-OUTPUT SECTION.                                                         0009
   8 FILE-CONTROL.                                                                 0010
   9     SELECT INP-FILE                                                           0011
             ASSIGN TO DISK-INPFILE.                                               0012
  10     SELECT MST-FILE                                                           0013
             ASSIGN TO DISK-MSTFILE.                                               0014
                                                                                   0015
  11 DATA DIVISION.                                                                0016
  12 FILE SECTION.                                                                 0017
  13 FD  INP-FILE                                                                  0018
         LABEL RECORDS ARE STANDARD.                                               0019
  14 01  INP-REC.                                                                 0020
  15     05  INP-RECTYPE              PICTURE XX.                                 0021
  16     05  INP-EMPLOYEE-NUMBER      PICTURE 9(6).                               0022
  17     05  INP-EMPLOYEE-NAME        PICTURE X(28).                              0023
  18     05  INP-EMPLOYEE-CODE        PICTURE 9.                                  0024
  19     05  INP-EMPLOYEE-SALARY      PICTURE 9(6)V99.                            0025
  20 FD  MST-FILE                                                                  0026
         LABEL RECORDS ARE STANDARD.                                               0027
  21 01  MST-REC.                                                                 0028
  22     05  MST-RECTYPE              PICTURE XX.                                 0029
  23     05  MST-EMPLOYEE-NUMBER      PICTURE 9(6).                               0030
  24     05  MST-EMPLOYEE-NAME        PICTURE X(28).                              0031
  25     05  MST-EMPLOYEE-CODE        PICTURE 9.                                  0032
  26     05  MST-EMPLOYEE-SALARY      PICTURE 9(6)V99.                            0033
  27 WORKING-STORAGE SECTION.                                                      0034
  28 01  THE-INPUTIND                 PICTURE X VALUE SPACE.                      0035
  29     88  THE-END-OF-INPUT         VALUE 'E'.                                  0036
  30 01  THE-MASTERIND                PICTURE X VALUE SPACE.                      0037
  31     88  THE-END-OF-MASTER        VALUE 'E'.                                  0038
  32 01  THE-MSTTYPE                  PICTURE XX VALUE 'M1'.                      0039
                                                                                   0040
  33 PROCEDURE DIVISION.                                                           0041
  34 TOP-LOGIC-PARA.                                                               0042
  35     OPEN INPUT INP-FILE                                                       0043
              I-O    MST-FILE.                                                     0044
  36     PERFORM READ-INPUT.                                                       0045
  37     PERFORM READ-MASTER.                                                      0046
  38     PERFORM PROCESS-PARA UNTIL THE-END-OF-INPUT.                              0047
  39     CLOSE INP-FILE                                                            0048
               MST-FILE.                                                           0049
  40     STOP RUN.                                                                 0050
```

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S ..........IDENTFCN SEQ/NO S

 41  READ-INPUT.                                                          0051
 42      READ INP-FILE                                                    0052
 43          AT END SET THE-END-OF-INPUT TO TRUE.                         0053
 44  READ-MASTER.                                                         0054
 45      READ MST-FILE                                                    0055
 46          AT END SET THE-END-OF-MASTER TO TRUE                         0056
 47                 CLOSE MST-FILE                                        0057
 48                 OPEN EXTEND MST-FILE.                                 0058
 49  PROCESS-PARA.                                                        0059
 50      IF THE-END-OF-MASTER                                             0060
 51         MOVE THE-MSTTYPE TO INP-RECTYPE                               0061
 52         WRITE MST-REC FROM INP-REC                                    0062
 53         PERFORM READ-INPUT                                            0063
 54      ELSE IF MST-EMPLOYEE-NUMBER LESS THAN INP-EMPLOYEE-NUMBER        0064
 56              PERFORM READ-MASTER                                      0065
 57           ELSE IF MST-EMPLOYEE-NUMBER = INP-EMPLOYEE-NUMBER           0066
 59                   MOVE THE-MSTTYPE TO INP-RECTYPE                     0067
 60                   REWRITE MST-REC FROM INP-REC                        0068
 61                   PERFORM READ-INPUT                                  0069
 62                   PERFORM READ-MASTER                                 0070
 63                ELSE DISPLAY 'ERROR RECORD-->' INP-EMPLOYEE-NUMBER.    0071
```

Indexed File Creation

This program creates an indexed file of summary records for bank depositors. The key within each indexed file record is OUT-RECKEY (the depositor's account number); the input records are ordered in ascending sequence upon this key. Records are read from the input file and transferred to the indexed file record area. The indexed file record is then written.

Note: The program-id used by the system is CRTIND.

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S .........IDENTFCN SEQ/NO S

     PROCESS SOURCE                                                          0001
   1 IDENTIFICATION DIVISION.                                                0002
   2 PROGRAM-ID.  CRTINDEXED.                                                0003
                                                                             0004
   3 ENVIRONMENT DIVISION.                                                   0005
   4 CONFIGURATION SECTION.                                                  0006
   5 SOURCE-COMPUTER.  IBM-S34.                                              0007
   6 OBJECT-COMPUTER.  IBM-S34.                                              0008
   7 INPUT-OUTPUT SECTION.                                                   0009
   8 FILE-CONTROL.                                                           0010
   9     SELECT INP-FILE                                                     0011
             ASSIGN TO DISK-INPFILE.                                         0012
  10     SELECT OUT-FILE                                                     0013
             ASSIGN TO DISK-MSTFILE                                          0014
             ACCESS IS SEQUENTIAL                                            0015
             ORGANIZATION IS INDEXED                                         0016
             RECORD KEY IS OUT-RECKEY                                        0017
             FILE STATUS IS OUT-STATUS.                                      0018
                                                                             0019
  11 DATA DIVISION.                                                          0020
  12 FILE SECTION.                                                           0021
  13 FD  INP-FILE                                                            0022
         LABEL RECORDS ARE STANDARD.                                         0023
  14 01  INP-REC.                                                           0024
  15     05  INP-RECKEY              PICTURE X(10).                         0025
  16     05  INP-NAME                PICTURE X(20).                         0026
  17     05  INP-BAL                 PICTURE S9(5)V99.                      0027
  18 FD  OUT-FILE                                                            0028
         LABEL RECORDS ARE STANDARD.                                         0029
  19 01  OUT-REC.                                                           0030
  20     05  OUT-RECKEY              PICTURE X(10).                         0031
  21     05  OUT-FLD1                PICTURE X(10).                         0032
  22     05  OUT-NAME                PICTURE X(20).                         0033
  23     05  OUT-BAL                 PICTURE S9(5)V99.                      0034
  24 WORKING-STORAGE SECTION.                                                0035
  25 01  THE-INPUTIND                PICTURE X VALUE SPACE.                 0036
  26     88  THE-END-OF-INPUT        VALUE 'E'.                             0037
  27 01  OUT-STATUS                  PICTURE XX.                            0038
                                                                             0039
  28 PROCEDURE DIVISION.                                                     0040
                                                                             0041
  29 TOP-LOGIC-PARA.                                                         0042
  30     OPEN INPUT  INP-FILE                                                0043
              OUTPUT OUT-FILE.                                               0044
  31     READ INP-FILE                                                       0045
  32         AT END SET THE-END-OF-INPUT TO TRUE.                            0046
  33     PERFORM PROCESS-DATA UNTIL THE-END-OF-INPUT.                        0047
  34     CLOSE INP-FILE                                                      0048
               OUT-FILE.                                                     0049
  35     STOP RUN.                                                           0050
  36 PROCESS-DATA.                                                           0051
  37     MOVE INP-RECKEY TO OUT-RECKEY.                                      0052
  38     MOVE INP-NAME   TO OUT-NAME.                                        0053
  39     MOVE INP-BAL    TO OUT-BAL.                                         0054
  40     MOVE SPACES     TO OUT-FLD1.                                        0055
  41     WRITE OUT-REC                                                       0056
  42         INVALID KEY DISPLAY 'WRITE FAILED FOR KEY ' OUT-RECKEY.         0057
  43     READ INP-FILE                                                       0058
  44         AT END SET THE-END-OF-INPUT TO TRUE.                            0059
                                                                             0060
```

Indexed File Updating

This program, using dynamic access, updates the indexed file created in the CRTINDEXED program.

The input records contain the key for the record, the depositor name, and the amount of the transaction.

When the input record is read, the program tests whether this is a transaction record (in which case, all fields of the record are filled) or a record requesting sequential retrieval of a specific generic class (in which case, only the UPD-GENFLD of the input record contains data).

Random access is used for the updating and printing of the transaction records. Sequential access is used for the retrieval and printing of all records within one generic class.

Note: The program-id used by the system is UPDTIN.

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S .........IDENTFCN SEQ/NO S

     PROCESS SOURCE                                                          0001
   1 IDENTIFICATION DIVISION.                                                0002
   2 PROGRAM-ID.  UPDTINDEXED.                                               0003
                                                                             0004
   3 ENVIRONMENT DIVISION.                                                   0005
   4 CONFIGURATION SECTION.                                                  0006
   5 SOURCE-COMPUTER.  IBM-S34.                                              0007
   6 OBJECT-COMPUTER.  IBM-S34.                                              0008
   7 INPUT-OUTPUT SECTION.                                                   0009
   8 FILE-CONTROL.                                                           0010
   9     SELECT MST-FILE                                                     0011
             ASSIGN TO DISK-MSTFILE                                          0012
             ACCESS IS DYNAMIC                                               0013
             ORGANIZATION IS INDEXED                                         0014
             RECORD KEY IS MST-KEY                                           0015
             FILE STATUS IS MST-STATUS.                                      0016
  10     SELECT UPD-FILE                                                     0017
             ASSIGN TO DISK-UPDFILE.                                         0018
  11     SELECT PRT-FILE                                                     0019
             ASSIGN TO PRINTER-PRTFILE.                                      0020
                                                                             0021
  12 DATA DIVISION.                                                          0022
  13 FILE SECTION.                                                           0023
  14 FD  MST-FILE                                                            0024
         LABEL RECORDS ARE STANDARD.                                         0025
  15 01  MST-REC.                                                           0026
  16     05  MST-KEY.                                                       0027
  17         10  MST-GENFLD          PICTURE X(5).                         0028
  18         10  MST-DETFLD          PICTURE X(5).                         0029
  19     05  MST-FLD1                PICTURE X(10).                        0030
  20     05  MST-NAME                PICTURE X(20).                        0031
  21     05  MST-BAL                 PICTURE S9(5)V99.                     0032
  22 FD  UPD-FILE                                                            0033
         LABEL RECORDS ARE STANDARD.                                         0034
  23 01  UPD-REC.                                                           0035
  24     05  UPD-KEY.                                                       0036
  25         10  UPD-GENFLD          PICTURE X(5).                         0037
  26         10  UPD-DETFLD          PICTURE X(5).                         0038
  27     05  UPD-NAME                PICTURE X(20).                        0039
  28     05  UPD-AMT                 PICTURE S9(5)V99.                     0040
  29 FD  PRT-FILE                                                            0041
         LABEL RECORDS ARE OMITTED                                           0042
         LINAGE IS 51 LINES WITH FOOTING AT 48.                              0043
  30 01  PRT-REC.                                                           0044
  31     05  PRT-KEY                 PICTURE X(10).                        0045
  32     05  PRT-NAME                PICTURE B(5)X(20).                    0046
  33     05  FILLER                  PICTURE X(5).                         0047
  34     05  PRT-BAL                 PICTURE $$$$.$$$.99-.                 0048
  35     05  FILLER                  PICTURE X(5).                         0049
  36     05  PRT-AMT                 PICTURE $$$$.$$$.99-.                 0050
```

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S .........IDENTFCN SEQ/NO S

 37       05  FILLER                  PICTURE X(5).                          0051
 38       05  PRT-NEWBAL              PICTURE $$$$,$$$.99-.                  0052
 39   WORKING-STORAGE SECTION.                                               0053
 40   01  PAGE-HEAD.                                                         0054
 41       05  FILLER                  PICTURE X(38) VALUE SPACES.            0055
 42       05  FILLER                  PICTURE X(13)                          0056
                                       VALUE 'UPDATE REPORT'.                0057
 43       05  FILLER                  PICTURE X(38) VALUE SPACES.            0058
 44   01  PAGE-FOOT.                                                         0059
 45       05  FILLER                  PICTURE X(81) VALUE SPACES.            0060
 46       05  FILLER                  PICTURE X(6)  VALUE 'PAGE #'.          0061
 47       05  PAGE-NUMBER             PICTURE 99    VALUE ZERO.              0062
 48   01  ERROR-MESSAGE.                                                     0063
 49       05  ERR-NAME                PICTURE X(7).                          0064
 50       05  FILLER                  PICTURE XX    VALUE SPACES.            0065
 51       05  MST-STATUS              PICTURE XX.                            0066
 52           88  MST-GDSTATUS          VALUE ZERO.                          0067
 53   01  THE-INDICATOR               PICTURE X VALUE SPACE.                 0068
 54       88  THE-END-OF-INPUT          VALUE 'E'.                           0069
 55   01  THE-OPNAMES.                                                       0070
 56       05  THE-READ-D              PICTURE X(7) VALUE 'READ-D '.          0071
 57       05  THE-START               PICTURE X(7) VALUE 'START  '.          0072
 58       05  THE-READ-S              PICTURE X(7) VALUE 'READ-S '.          0073
 59       05  THE-REWRITE             PICTURE X(7) VALUE 'REWRITE'.          0074
                                                                             0075
 60   PROCEDURE DIVISION.                                                    0076
                                                                             0077
 61   TOP-LOGIC-PARA.                                                        0078
 62       OPEN INPUT  UPD-FILE                                               0079
              I-O    MST-FILE                                               0080
              OUTPUT PRT-FILE.                                               0081
 63       PERFORM PAGE-START.                                                0082
 64       READ UPD-FILE                                                      0083
 65           AT END SET THE-END-OF-INPUT TO TRUE.                           0084
 66       PERFORM PROCESS-DATA UNTIL THE-END-OF-INPUT.                       0085
 67       CLOSE UPD-FILE                                                     0086
                MST-FILE                                                    0087
                PRT-FILE.                                                   0088
 68       STOP RUN.                                                          0089
                                                                             0090
 69   PROCESS-DATA.                                                          0091
 70       IF UPD-DETFLD EQUAL SPACES                                         0092
 71           PERFORM SEQUENTIAL-PROCESS                                     0093
 72       ELSE PERFORM DYNAMIC-PROCESS.                                      0094
 74       READ UPD-FILE                                                      0095
 75           AT END SET THE-END-OF-INPUT TO TRUE.                           0096
                                                                             0097
 76   SEQUENTIAL-PROCESS.                                                    0098
 77       MOVE UPD-GENFLD TO MST-GENFLD                                      0099
 78       START MST-FILE KEY = MST-GENFLD                                    0100
 79           INVALID KEY MOVE HIGH-VALUE TO MST-GENFLD.                     0101
```

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S ..........IDENTFCN SEQ/NO S

 80       IF NOT MST-GDSTATUS                                          0102
 81           MOVE THE-START TO ERR-NAME                               0103
 82           DISPLAY ERROR-MESSAGE.                                   0104
 83       PERFORM SEQUENTIAL-PROCESS-A10                               0105
              UNTIL UPD-GENFLD NOT EQUAL MST-GENFLD.                   0106
 84   SEQUENTIAL-PROCESS-A10.                                          0107
 85       READ MST-FILE NEXT RECORD                                    0108
 86           AT END MOVE HIGH-VALUE TO MST-GENFLD.                    0109
 87       IF UPD-GENFLD EQUAL MST-GENFLD                               0110
 88           MOVE MST-KEY  TO PRT-KEY                                 0111
 89           MOVE MST-NAME TO PRT-NAME                                0112
 90           MOVE MST-BAL  TO PRT-NEWBAL                              0113
 91           PERFORM PRINT-DETAIL.                                    0114
                                                                       0115
 92   DYNAMIC-PROCESS.                                                 0116
 93       MOVE UPD-KEY TO MST-KEY.                                     0117
 94       READ MST-FILE                                                0118
 95           INVALID KEY MOVE HIGH-VALUE TO MST-KEY.                  0119
 96       IF NOT MST-GDSTATUS                                          0120
 97           MOVE THE-READ-D TO ERR-NAME                              0121
 98           DISPLAY ERROR-MESSAGE                                    0122
 99           SET MST-GDSTATUS TO TRUE                                 0123
100       ELSE MOVE MST-KEY  TO PRT-KEY                                0124
102           MOVE MST-NAME  TO PRT-NAME                               0125
103           MOVE MST-BAL   TO PRT-BAL                                0126
104           MOVE UPD-AMT   TO PRT-AMT                                0127
105           ADD UPD-AMT    TO MST-BAL                                0128
106           MOVE MST-BAL   TO PRT-NEWBAL                             0129
107           PERFORM PRINT-DETAIL                                     0130
108           REWRITE MST-REC                                          0131
109               INVALID KEY MOVE HIGH-VALUE TO MST-KEY.              0132
110       IF NOT MST-GDSTATUS                                          0133
111           MOVE THE-REWRITE TO ERR-NAME                             0134
112           DISPLAY ERROR-MESSAGE.                                   0135
                                                                       0136
113   PRINT-DETAIL.                                                    0137
114       WRITE PRT-REC                                                0138
115           AT END-OF-PAGE PERFORM PAGE-END                          0139
                     THRU PAGE-START.                                  0140
116       MOVE SPACES TO PRT-REC.                                      0141
117   PAGE-END.                                                        0142
118       ADD 1 TO PAGE-NUMBER.                                        0143
119       WRITE PRT-REC FROM PAGE-FOOT AFTER ADVANCING 3.              0144
120   PAGE-START.                                                      0145
121       WRITE PRT-REC FROM PAGE-HEAD AFTER ADVANCING PAGE.           0146
122       MOVE SPACES TO PRT-REC.                                      0147
```

Relative File Creation

This program creates a relative file of summary sales records using sequential access. Each record contains a 5-year summary of unit and dollar sales for one week of the year; there are 51 records within the file, each representing one week.

Each input record represents the summary sales for one week of one year. The records for the first week of the last five years (in ascending order) are the first five input records. The records for the second week of the last five years are the next five input records,and so forth. Thus, five input records fill one output record.

The RELATIVE KEY for the OUT-FILE is not specified because it is not required for sequential access unless the START statement is used. (For updating, however, the key is INP-WEEK.)

Note: The program-id used by the system is CRTREL.

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S .........IDENTFCN SEQ/NO S

     PROCESS SOURCE                                                          0001
   1 IDENTIFICATION DIVISION.                                                0002
   2 PROGRAM-ID.  CRTRELATIVE.                                               0003
                                                                             0004
   3 ENVIRONMENT DIVISION.                                                   0005
   4 CONFIGURATION SECTION.                                                  0006
   5 SOURCE-COMPUTER.  IBM-S34.                                              0007
   6 OBJECT-COMPUTER.  IBM-S34.                                              0008
   7 INPUT-OUTPUT SECTION.                                                   0009
   8 FILE-CONTROL.                                                           0010
   9     SELECT INP-FILE                                                     0011
             ASSIGN TO DISK-INPFILE.                                         0012
  10     SELECT OUT-FILE                                                     0013
             ASSIGN TO DISK-SUMFILE                                          0014
             ACCESS IS SEQUENTIAL                                            0015
             ORGANIZATION IS RELATIVE                                        0016
             FILE STATUS IS OUT-STATUS.                                      0017
                                                                             0018
  11 DATA DIVISION.                                                          0019
  12 FILE SECTION.                                                           0020
  13 FD  INP-FILE                                                            0021
         LABEL RECORDS ARE STANDARD.                                         0022
  14 01  INP-REC.                                                           0023
  15     05  INP-YEAR                 PICTURE 99.                           0024
  16     05  INP-WEEK                 PICTURE 99.                           0025
  17     05  INP-UNIT-SALES           PICTURE S9(6).                        0026
  18     05  INP-DOLLAR-SALES         PICTURE S9(9)V99.                     0027
  19 FD  OUT-FILE                                                            0028
         LABEL RECORDS ARE STANDARD.                                         0029
  20 01  OUT-REC.                                                           0030
  21     05  OUT-DATA OCCURS 5 TIMES INDEXED BY OUT-IND.                    0031
  22         10  OUT-YEAR             PICTURE 99.                           0032
  23         10  OUT-WEEK             PICTURE 99.                           0033
  24         10  OUT-UNIT-SALES       PICTURE S9(6).                        0034
  25         10  OUT-DOLLAR-SALES     PICTURE S9(9)V99.                     0035
  26 WORKING-STORAGE SECTION.                                                0036
  27 01  THE-INPUTIND                 PICTURE X VALUE SPACE.                0037
  28     88  THE-END-OF-INPUT           VALUE 'E'.                          0038
  29 01  OUT-STATUS                   PICTURE XX.                           0039
  30 PROCEDURE DIVISION.                                                     0040
  31 TOP-LOGIC-PARA.                                                         0041
  32     OPEN INPUT  INP-FILE                                                0042
              OUTPUT OUT-FILE.                                               0043
  33     SET OUT-IND TO 1.                                                   0044
  34     READ INP-FILE                                                       0045
  35         AT END SET THE-END-OF-INPUT TO TRUE.                            0046
  36     PERFORM PROCESS-DATA UNTIL THE-END-OF-INPUT.                        0047
  37     CLOSE INP-FILE                                                      0048
               OUT-FILE.                                                     0049
  38     STOP RUN.                                                           0050
  39 PROCESS-DATA.                                                           0051
  40     MOVE INP-REC TO OUT-DATA (OUT-IND).                                 0052
  41     IF OUT-IND NOT = 5                                                  0053
  42         SET OUT-IND UP BY 1                                             0054
  43     ELSE SET OUT-IND TO 1                                               0055
  45         WRITE OUT-REC                                                   0056
  46             INVALID KEY DISPLAY 'WRITE FAILED' OUT-STATUS.              0057
  47     READ INP-FILE                                                       0058
  48         AT END SET THE-END-OF-INPUT TO TRUE.                            0059
```

Relative File Updating

This program uses sequential access to update the file of summary sales records created in the CRTRELATIVE program. The updating program adds a record for the new year and deletes the oldest year's records from the MST-FILE.

The input record represents the summary sales record for one week of the preceding year. The RELATIVE KEY for the SUM-FILE is present in the input record as UPD-WEEK. The RELATIVE KEY is used to check that the record was correctly written.

Note: The program-id used by the system is UPDTRE.

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S .........IDENTFCN SEQ/NO S

     PROCESS SOURCE                                                          0001
   1 IDENTIFICATION DIVISION.                                                0002
   2 PROGRAM-ID.  UPDTRELATIVE.                                              0003
                                                                             0004
   3 ENVIRONMENT DIVISION.                                                   0005
   4 CONFIGURATION SECTION.                                                  0006
   5 SOURCE-COMPUTER.  IBM-S34.                                              0007
   6 OBJECT-COMPUTER.  IBM-S34.                                              0008
   7 INPUT-OUTPUT SECTION.                                                   0009
   8 FILE-CONTROL.                                                           0010
   9     SELECT SUM-FILE                                                     0011
             ASSIGN TO DISK-SUMFILE                                          0012
             ACCESS IS SEQUENTIAL                                            0013
             ORGANIZATION IS RELATIVE                                        0014
             FILE STATUS IS SUM-STATUS.                                      0015
  10     SELECT UPD-FILE                                                     0016
             ASSIGN TO DISK-UPDFILE.                                         0017
                                                                             0018
  11 DATA DIVISION.                                                          0019
  12 FILE SECTION.                                                           0020
  13 FD  SUM-FILE                                                            0021
         LABEL RECORDS ARE STANDARD.                                         0022
  14 01  SUM-REC                        PICTURE X(105).                     0023
  15 FD  UPD-FILE                                                            0024
         LABEL RECORDS ARE STANDARD.                                         0025
  16 01  UPD-REC.                                                           0026
  17     05  UPD-YEAR                   PICTURE 99.                         0027
  18     05  UPD-WEEK                   PICTURE 99.                         0028
  19     05  UPD-UNIT-SALES             PICTURE S9(6).                      0029
  20     05  UPD-DOLLAR-SALES           PICTURE S9(9)V99.                   0030
  21 WORKING-STORAGE SECTION.                                                0031
  22 01  WORK-RECORD.                                                       0032
  23     02  FILLER                     PICTURE X(21).                      0033
  24     02  OUT-REC.                                                       0034
  25         05  FILLER                 PICTURE X(84).                      0035
  26         05  WORK-INFO.                                                 0036
  27           10  WORK-YEAR            PICTURE 99.                         0037
  28           10  WORK-WEEK            PICTURE 99.                         0038
  29           10  WORK-UNITSALES       PICTURE S9(6).                      0039
  30           10  WORK-DOLLAR-SALES    PICTURE S9(9)V99.                   0040
  31 01  THE-INDICATOR                  PICTURE X VALUE SPACE.              0041
  32     88  THE-END-OF-INPUT             VALUE 'E'.                        0042
  33 01  ERROR-MESSAGE.                                                     0043
  34     05  FILLER                     PICTURE X(30) VALUE                 0044
         'REWRITE ERROR - FILE STATUS = '.                                   0045
  35     05  SUM-STATUS                 PICTURE XX.                         0046
                                                                             0047
  36 PROCEDURE DIVISION.                                                     0048
  37 TOP-LOGIC-PARA.                                                         0049
  38     OPEN INPUT UPD-FILE                                                 0050
              I-O   SUM-FILE.                                                0051
  39     READ SUM-FILE INTO WORK-RECORD                                      0052
  40         AT END SET THE-END-OF-INPUT TO TRUE.                            0053
  41     READ UPD-FILE INTO WORK-INFO                                        0054
  42         AT END SET THE-END-OF-INPUT TO TRUE.                            0055
  43     PERFORM PROCESS-PARA UNTIL THE-END-OF-INPUT.                        0056
  44     CLOSE UPD-FILE                                                      0057
               SUM-FILE.                                                     0058
  45     STOP RUN.                                                           0059
  46 PROCESS-PARA.                                                           0060
  47     REWRITE SUM-REC FROM OUT-REC.                                       0061
  48     IF SUM-STATUS NOT = ZERO                                            0062
  49         DISPLAY ERROR-MESSAGE.                                          0063
  50     READ SUM-FILE INTO WORK-RECORD                                      0064
  51         AT END SET THE-END-OF-INPUT TO TRUE.                            0065
  52     READ UPD-FILE INTO WORK-INFO                                        0066
  53         AT END SET THE-END-OF-INPUT TO TRUE.                            0067
```

Relative File Retrieval

This program, using dynamic access, retrieves the summary file created by the CRTRELATIVE program.

The records of the INP-FILE contain one required field (INP-WEEK), which is the RELATIVE KEY for RELATIVE-FILE, and one optional field (INP-END-WEEK). An input record containing data in INP-WEEK and spaces in INP-END-WEEK requests a printout for that one specific SUM-REC; the record is retrieved through random access. An input record containing data in both INP-WEEK and INP-END-WEEK requests a printout of all the SUM-FILE records within the RELATIVE KEY range of INP-WEEK through INP-END-WEEK, inclusive; these records are retrieved through sequential access.

Note: The program-id used by the system is RTRVRE.

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S .........IDENTFCN SEQ/NO S

     PROCESS SOURCE                                                          0001
   1 IDENTIFICATION DIVISION.                                                0002
   2 PROGRAM-ID.  RTRVRELATIVE.                                              0003
                                                                             0004
   3 ENVIRONMENT DIVISION.                                                   0005
   4 CONFIGURATION SECTION.                                                  0006
   5 SOURCE-COMPUTER.  IBM-S34.                                              0007
   6 OBJECT-COMPUTER.  IBM-S34.                                              0008
   7 INPUT-OUTPUT SECTION.                                                   0009
   8 FILE-CONTROL.                                                           0010
   9     SELECT SUM-FILE                                                     0011
             ASSIGN TO DISK-SUMFILE                                          0012
             ACCESS IS DYNAMIC                                               0013
             ORGANIZATION IS RELATIVE                                        0014
             RELATIVE KEY IS INP-WEEK                                        0015
             FILE STATUS IS SUM-STATUS.                                      0016
  10     SELECT INP-FILE                                                     0017
          .  ASSIGN TO DISK-INPFILE.                                         0018
  11     SELECT PRT-FILE                                                     0019
             ASSIGN TO PRINTER-PRTFILE.                                      0020
  12 DATA DIVISION.                                                         0021
  13 FILE SECTION.                                                          0022
  14 FD  SUM-FILE                                                           0023
         LABEL RECORDS ARE STANDARD.                                        0024
  15 01  SUM-REC.                                                          0025
  16     05  SUM-DATA OCCURS 5 TIMES INDEXED BY SUM.                        0026
  17         10  SUM-YEAR             PICTURE 99.                         0027
  18         10  SUM-WEEK             PICTURE 99.                         0028
  19         10  SUM-UNIT-SALES       PICTURE S9(6).                      0029
  20         10  SUM-DOLLAR-SALES     PICTURE S9(9)V99.                   0030
  21 FD  INP-FILE                                                           0031
         LABEL RECORDS ARE STANDARD.                                        0032
  22 01  INP-REC.                                                          0033
  23     05  INP-WEEK                 PICTURE 99.                         0034
  24     05  INP-END-WEEK             PICTURE 99.                         0035
  25 FD  PRT-FILE                                                           0036
         LABEL RECORDS ARE OMITTED.                                         0037
  26 01  PRT-REC.                                                          0038
  27     05  PRT-WEEK                 PICTURE 99.                         0039
  28     05  PRT-YEAR                 PICTURE B(5)99.                     0040
  29     05  PRT-UNIT-SALES            PICTURE B(5)99.                    0041
  30     05  FILLER                   PICTURE X(5).                       0042
  31     05  PRT-DOLLAR-SALES          PICTURE $$$$.$$$.$$$.99.           0043
  32 WORKING-STORAGE SECTION.                                               0044
  33 01  THE-INDICATOR                PICTURE X VALUE SPACE.              0045
  34     88  THE-END-OF-INPUT           VALUE 'E'.                        0046
  35 01  SUM-STATUS                   PICTURE XX.                         0047
                                                                             0048
  36 PROCEDURE DIVISION.                                                    0049
  37 TOP-LOGIC-PARA.                                                        0050
```

```
STNO -A...B... C O B O L   S O U R C E   S T A T E M E N T S ..........IDENTFCN SEQ/NO S

 38       OPEN INPUT  SUM-FILE INP-FILE                                    0051
               OUTPUT PRT-FILE.                                            0052
 39       READ INP-FILE                                                    0053
 40           AT END SET THE-END-OF-INPUT TO TRUE.                         0054
 41       PERFORM PROCESS-INPUT UNTIL THE-END-OF-INPUT.                    0055
 42       CLOSE SUM-FILE                                                   0056
                INP-FILE                                                  0057
                PRT-FILE.                                                 0058
 43       STOP RUN.                                                        0059
 44   PROCESS-INPUT.                                                       0060
 45       IF INP-END-WEEK EQUAL SPACES                                     0061
 46           PERFORM RANDOM-PROCESS                                       0062
 47       ELSE PERFORM SEQUENTIAL-PROCESS.                                 0063
 49       READ INP-FILE                                                    0064
 50           AT END SET THE-END-OF-INPUT TO TRUE.                         0065
 51   RANDOM-PROCESS.                                                      0066
 52       READ SUM-FILE                                                    0067
 53           INVALID KEY MOVE HIGH-VALUE TO SUM-YEAR (1).                 0068
 54       IF SUM-YEAR (1) NOT EQUAL HIGH-VALUE                             0069
 55           PERFORM PRINT-SUMMARY.                                       0070
 56   SEQUENTIAL-PROCESS.                                                  0071
 57       READ SUM-FILE                                                    0072
 58           INVALID KEY MOVE HIGH-VALUE TO SUM-YEAR (1).                 0073
 59       PERFORM SEQUENTIAL-PROCESS-A10                                   0074
               UNTIL SUM-YEAR (1) GREATER THAN INP-END-WEEK.               0075
 60   SEQUENTIAL-PROCESS-A10.                                              0076
 61       PERFORM PRINT-SUMMARY.                                           0077
 62       READ SUM-FILE NEXT RECORD                                        0078
 63           AT END MOVE HIGH-VALUE TO SUM-YEAR (1).                      0079
 64   PRINT-SUMMARY.                                                       0080
 65       PERFORM PRINT-SUMMARY-A10 VARYING SUM FROM 1 BY 1                0081
               UNTIL SUM > 5.                                              0082
 66   PRINT-SUMMARY-A10.                                                   0083
 67       MOVE SUM-YEAR         (SUM) TO PRT-YEAR.                         0084
 68       MOVE SUM-WEEK         (SUM) TO PRT-WEEK.                         0085
 69       MOVE SUM-UNIT-SALES   (SUM) TO PRT-UNIT-SALES.                   0086
 70       MOVE SUM-DOLLAR-SALES (SUM) TO PRT-DOLLAR-SALES.                 0087
 71       WRITE PRT-REC AFTER ADVANCING 2.                                 0088
```

COBOL Sort Example

This program prints a report from the sorted output of two disk files. The INPUT-FILE and the INPUT-FILE2 are merged together into the SORT-FILE and then sorted in ascending sequence on SORT-FIELD2 and descending sequence on SORT-FIELD1.

The OUTPUT PROCEDURE option of the SORT statement is used to print the report. Note that the sort input files must not be OPEN and must be described as sequential organization. The System/34 will allow indexed and relative files to be described as sequential input only files. Note also that the OUTPUT PROCEDURE must reference a SECTION.

```
00010   PROCESS OBJECT,LET,LIST                                          SORT1
00020   IDENTIFICATION DIVISION.                                         SORT1
00030   PROGRAM-ID. SORT1.                                               SORT1
00040   INSTALLATION. ROCHESTER MARKET SUPPORT CENTER,                   SORT1
00050       SAMPLE COBOL SORT PROGRAM, SORTS TWO INPUT FILES AND PRINTS  SORT1
00060       A REPORT USING THE OUTPUT PROCEDURE OPTION.                  SORT1
00070                                                                    SORT1
00080   ENVIRONMENT DIVISION.                                            SORT1
00090   CONFIGURATION SECTION.                                           SORT1
00100   SOURCE-COMPUTER. IBM-S34.                                        SORT1
00110   OBJECT-COMPUTER. IBM-S34.                                        SORT1
00120   INPUT-OUTPUT SECTION.                                            SORT1
00130   FILE-CONTROL.                                                    SORT1
00140       SELECT SORT-FILE   ASSIGN TO DISK-SORTWORK.                  SORT1
00150       SELECT INPUT-FILE  ASSIGN TO DISK-ITEMMAST.                  SORT1
00160       SELECT INPUT-FILE2 ASSIGN TO DISK-OLDITEMS.                  SORT1
00170       SELECT PRINT-FILE  ASSIGN TO PRINTER-COBPNT01.               SORT1
00180                                                                    SORT1
00190   DATA DIVISION.                                                   SORT1
00200   FILE SECTION.                                                    SORT1
00210   SD  SORT-FILE.                                                   SORT1
00220   01  SORT-REC.                                                    SORT1
00230       03  SORT-FIELD1   PIC XXXX.                                 SORT1
00240       03  FILLER        PIC X(124).                               SORT1
00250       03  SORT-FIELD2   PIC 9999.                                 SORT1
00260   FD  INPUT-FILE   LABEL RECORDS ARE STANDARD.                     SORT1
00270   01  FILE-REC          PIC X(132).                               SORT1
00280   FD  INPUT-FILE2  LABEL RECORDS ARE STANDARD.                     SORT1
00290   01  FILE-REC2         PIC X(132).                               SORT1
00300   FD  PRINT-FILE   LABEL RECORDS ARE OMITTED.                      SORT1
00310   01  PRINT-REC         PIC X(132).                               SORT1
00320   WORKING-STORAGE SECTION.                                         SORT1
00330                                                                    SORT1
00340   PROCEDURE DIVISION.                                              SORT1
00350   START-PROGRAM SECTION.                                           SORT1
00360   MAIN-FUNCTION.                                                   SORT1
00370       OPEN OUTPUT PRINT-FILE.                                      SORT1
00380       SORT SORT-FILE ASCENDING KEY SORT-FIELD2                     SORT1
00390           ON DESCENDING KEY SORT-FIELD1                            SORT1
00400           USING INPUT-FILE, INPUT-FILE2                            SORT1
00410           OUTPUT PROCEDURE IS PRINT-REPORT.                        SORT1
00420       CLOSE PRINT-FILE.                                            SORT1
00430       STOP RUN.                                                    SORT1
00440   PRINT-REPORT SECTION.                                            SORT1
00450   PRINT-FUNCTION.                                                  SORT1
00460       RETURN SORT-FILE INTO PRINT-REC AT END GO TO FINISHED.       SORT1
00470       WRITE PRINT-REC AFTER ADVANCING 1 LINE.                      SORT1
00480       GO TO PRINT-FUNCTION                                         SORT1
00490   FINISHED.  EXIT.                                                 SORT1
```

Appendix F. IBM American National Standard COBOL Reserved Words

No word in the following two lists should appear as a programmer-defined name.

RESERVED WORDS USED BY THE SYSTEM/34 COBOL COMPILER

Words preceded by an asterisk (*) are not included in the American National Standard COBOL, X3.23-1974, reserved word list.

ACCEPT
ACCESS
*ACQUIRE
ADD
ADVANCING
AFTER
ALL
ALPHABETIC
ALSO
ALTER
ALTERNATE
AND
*APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
*ATTRIBUTE-DATA
AUTHOR

BEFORE
BLANK
BLOCK
BOTTOM
BY

CALL
*CHANGED
CHARACTER
CHARACTERS
CLOSE
CODE-SET
COLLATING
COMMA
COMP
*COMP-3
*COMP-4
COMPUTATIONAL
*COMPUTATIONAL-3
*COMPUTATIONAL-4
COMPUTE
CONFIGURATION
CONTAINS
*CONTROL-AREA
COPY
CORR
CORRESPONDING
COUNT
*CORE-INDEX
*CSP
CURRENCY
*C01

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3

DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DISPLAY
DIVIDE
DIVISION
DOWN
*DROP
DYNAMIC

ELSE
END
END-OF-PAGE
ENTER
ENVIRONMENT
EOP
EQUAL
ERROR
EVERY
EXCEPTION
*EXHIBIT
EXIT
EXTEND

FD
FILE
FILE-CONTROL
FILLER
FIRST
FOOTING
FOR
*FORMAT
FROM

GIVING
GO
GREATER

HIGH-VALUE
HIGH-VALUES

IDENTIFICATION
IF
I-O
I-O-CONTROL
IN
INDEX
INDEXED
*INDIC
*INDICATOR
*INDICATORS
INITIAL
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS

JUST
JUSTIFIED

KEY

LABEL
LEADING
LEFT
LESS
LINAGE
LINAGE-COUNTER
LINE
LINES
LINKAGE
*LOCAL-DATA
LOCK
LOW-VALUE
LOW-VALUES

MEMORY
MERGE
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY

*NAMED
NATIVE
NEGATIVE
NEXT
NO
NOT
NUMERIC

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORGANIZATION
OUTPUT
OVERFLOW

PAGE
PERFORM
PIC
PICTURE
PLUS
POINTER
POSITIVE
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID

QUOTE
QUOTES

RANDOM
READ
RECORD
RECORDS
REDEFINES
REEL
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
*REQUESTOR
RERUN
RESERVE
RESET
RETURN
REVERSED
REWIND
REWRITE
RIGHT
*ROLLING
ROUNDED
RUN

SAME
SD
SEARCH
SECTION
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
START
*STARTING
STATUS
STOP
STRING
SUBTRACT
SYNC
SYNCHRONIZED
*SYSTEM-CONSOLE
*SYSTEM-SHUTDOWN

TALLYING
TERMINAL
THAN
*THEN
THROUGH
THRU
TIME
TIMES
TO
TOP
*TRACE
TRAILING
*TRANSACTION
*TRUE

UNIT
UNSTRING
UNTIL
UP
UPON
*UPSI-0
*UPSI-1
*UPSI-2
*UPSI-3
*UPSI-4
*UPSI-5
*UPSI-6
*UPSI-7
USAGE
USE
USING

VALUE
VALUES
VARYING

WHEN
WITH
WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

RESERVED WORDS NOT USED BY THE SYSTEM/34 COBOL COMPILER

The reserved words in the following list are not used by the System/34 COBOL compiler and should not be used if compatibility with other American National Standard COBOL compilers and CODASYL COBOL is desired.

CANCEL
CD
CF
CH
CLOCK-UNITS
COBOL
CODE
COLUMN
COMMUNICATION
CONTROL
CONTROLS

DE
DESTINATION
DETAIL
DISABLE
DUPLICATES

EGI
EMI
ENABLE
ESI

FINAL

GENERATE
GROUP

HEADING

INDICATE
INITIATE

LAST
LENGTH
LIMIT
LIMITS
LINE-COUNTER

MESSAGE

NUMBER

PAGE-COUNTER
PF
PH
POSITION
PRINTING

QUEUE

RD
RECEIVE
REFERENCES
REPORT
REPORTING
REPORTS
RF
RH

SEND
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUM
SUPPRESS
SYMBOLIC

TABLE
TAPE
TERMINATE
TEXT
TYPE

Appendix G. EBCDIC and ASCII Collating Sequences

The ascending collating sequences for both the EBCDIC (Extended Binary Coded Decimal Interchange Code) and ASCII (American National Standard [X3.4-1977] Code for Information Interchange) character sets are given in this appendix. Decimal positions within the sequence are given, as well as the binary representation, symbol, meaning for each character, and corresponding decimal position within the other sequence.

Note: When you are using the literal option of the alphabet-name clause, you must add 1 to the number shown in this appendix to specify the corresponding character. (The numbers in this appendix run from 0 through 255; the numbers in the literal option run from 1 through 256.)

EBCDIC COLLATING SEQUENCE

Collating Sequence	Bit Configuration	Symbol	Meaning	Corresponding ASCII Collating Sequence Number
0	00000000			0
.				
.				
64	01000000	SP	Space	32
.				
.				
74	01001010	¢	Cent sign	91
75	01001011	.	Period, decimal point	46
76	01001100	<	Less-than sign	60
77	01001101	(	Left parenthesis	40
78	01001110	+	Plus sign	43
79	01001111	|	Vertical bar, logical OR	33
80	01010000	&	Ampersand	38
.				
.				
90	01011010	!	Exclamation point	93
91	01011011	$	Dollar sign	36
92	01011100	*	Asterisk	42
93	01011101	)	Right parenthesis	41
94	01011110	;	Semicolon	59
95	01011111	¬	Logical NOT	94
96	01100000	-	Minus, hyphen	45
97	01100001	/	Slash	47
.				
.				
106	01101010	|	Broken vertical bar	124
107	01101011	,	Comma	44
108	01101100	%	Percent sign	37
109	01101101	_	Underscore	95
110	01101110	>	Greater-than sign	62
111	01101111	?	Question mark	63
.				
.				

Collating Sequence	Bit Configuration	Symbol	Meaning	Corresponding ASCII Collating Sequence Number
121	01111001	`	Grave accent	96
122	01111010	:	Colon	58
123	01111011	#	Number sign	35
124	01111100	@	At sign	64
125	01111101	'	Apostrophe, prime	39
126	01111110	=	Equal sign	61
127	01111111	"	Quotation mark	34
.				
129	10000001	a		97
130	10000010	b		98
131	10000011	c		99
132	10000100	d		100
133	10000101	e		101
134	10000110	f		102
135	10000111	g		103
136	10001000	h		104
137	10001001	i		105
.				
.				
145	10010001	j		106
146	10010010	k		107
147	10010011	l		108
148	10010100	m		109
149	10010101	n		110
150	10010110	o		111
151	10010111	p		112
152	10011000	q		113
153	10011001	r		114
.				
.				
161	10100001	~	Tilde	126
162	10100010	s		115
163	10100011	t		116
164	10100100	u		117
165	10100101	v		118
166	10100110	w		119
167	10100111	x		120
168	10101000	y		121
169	10101001	z		122
.				
.				

Collating Sequence	Bit Configuration	Symbol	Meaning	Corresponding ASCII Collating Sequence Number
192	11000000	{	Left brace	123
193	11000001	A		65
194	11000010	B		66
195	11000011	C		67
196	11000100	D		68
197	11000101	E		69
198	11000110	F		70
199	11000111	G		71
200	11001000	H		72
201	11001001	I		73
.				
.				
208	11010000	}	Right brace	125
209	11010001	J		74
210	11010010	K		75
211	11010011	L		76
212	11010100	M		77
213	11010101	N		78
214	11010110	O		79
215	11010111	P		80
216	11011000	Q		81
217	11011001	R		82
.				
.				
224	11100000	\	Reverse slant	92
.				
.				
226	11100010	S		83
227	11100011	T		84
228	11100100	U		85
229	11100101	V		86
230	11100110	W		87
231	11100111	X		88
232	11101000	Y		89
233	11101001	Z		90
.				
.				
240	11110000	0		48
241	11110001	1		49
242	11110010	2		50
243	11110011	3		51
244	11110100	4		52
245	11110101	5		53
246	11110110	6		54
247	11110111	7		55
248	11111000	8		56
249	11111001	9		57
.				
.				
.				
255				

ASCII COLLATING SEQUENCE

Collating Sequence	Bit Configuration	Symbol	Meaning	Corresponding EBCDIC Collating Sequence Number
0	00000000		Null	0
.				
.				
.				
32	00100000	SP	Space	64
33	00100001	!	Exclamation point[1]	79
34	00100010	"	Quotation mark	127
35	00100011	#	Number sign	123
36	00100100	$	Dollar sign	91
37	00100101	%	Percent sign	108
38	00100110	&	Ampersand	80
39	00100111	'	Apostrophe, prime	125
40	00101000	(	Left parenthesis	77
41	00101001	)	Right parenthesis	93
42	00101010	*	Asterisk	92
43	00101011	+	Plus sign	78
44	00101100	,	Comma	107
45	00101101	-	Minus, Hyphen	96
46	00101110	.	Period, decimal point	75
47	00101111	/	Slash	97
48	00110000	0		240
49	00110001	1		241
50	00110010	2		242
51	00110011	3		243
52	00110100	4		244
53	00110101	5		245
54	00110110	6		246
55	00110111	7		247
56	00111000	8		248
57	00111001	9		249
58	00111010	:	Colon	122
59	00111011	;	Semicolon	94
60	00111100	<	Less-than sign	76
61	00111101	=	Equal sign	126
62	00111110	>	Greater-than sign	110
63	00111111	?	Question mark	111
64	01000000	@	At sign	124

[1]The corresponding EBCDIC symbol is | (logical OR).

Collating Sequence	Bit Configuration	Symbol	Meaning	Corresponding EBCDIC Collating Sequence Number
65	01000001	A		193
66	01000010	B		194
67	01000011	C		195
68	01000100	D		196
69	01000101	E		197
70	01000110	F		198
71	01000111	G		199
72	01001000	H		200
73	01001001	I		201
74	01001010	J		209
75	01001011	K		210
76	01001100	L		211
77	01001101	M		212
78	01001110	N		213
79	01001111	O		214
80	01010000	P		215
81	01010001	Q		216
82	01010010	R		217
83	01010011	S		226
84	01010100	T		227
85	01010101	U		228
86	01010110	V		229
87	01010111	W		230
88	01011000	X		231
89	01011001	Y		232
90	01011010	Z		233
91	01011011	[	Opening bracket[1]	74
92	01011100	\	Reverse slant	224
93	01011101	]	Closing bracket[2]	90
94	01011110	^	Circumflex,	95
		¬	Logical NOT	
95	01011111	_	Underscore	109
96	01100000	`	Grave accent	121

[1]The corresponding EBCDIC symbol is ¢ (cent sign).
[2]The corresponding EBCDIC symbol is ! (exclamation point).

Collating Sequence	Bit Configuration	Symbol	Meaning	Corresponding EBCDIC Collating Sequence Number
97	01100001	a		129
98	01100010	b		130
99	01100011	c		131
100	01100100	d		132
101	01100101	e		133
102	01100110	f		134
103	01100111	g		135
104	01101000	h		136
105	01101001	i		137
106	01101010	j		145
107	01101011	k		146
108	01101100	l		147
109	01101101	m		148
110	01101110	n		149
111	01101111	o		150
112	01110000	p		151
113	01110001	q		152
114	01110010	r		153
115	01110011	s		162
116	01110100	t		163
117	01110101	u		164
118	01110110	v		165
119	01110111	w		166
120	01111000	x		167
121	01111001	y		168
122	01111010	z		169
123	01111011	{	Left Brace	192
124	01111100	|	Vertical line[1]	106
125	01111101	}	Right Brace	208
126	01111110	~	Tilde	161

[1]The corresponding EBCDIC symbol is ¦ (broken vertical line).

Appendix H. File Processing Summary and Status Key Values

This appendix illustrates, through the use of the first four figures, the various required and optional entries used with sequential, relative, indexed, and TRANSACTION files. Figure H-5 contains status key values and their meanings.

		Required Entries				
		Environment Division		Data Division	Procedure Division	
Device	**Function**	**SELECT**	**ASSIGN system-name**	**FD Entry**	**OPEN**	**CLOSE**
Printers	Print	file-name	PRINTER-external-name	file-name LABEL RECORDS OMITTED	OUTPUT file-name	file-name [WITH LOCK]
Disk	Read	file-name	DISK-external-name	file-name LABEL RECORDS STANDARD	INPUT file-name	file-name [WITH LOCK]
	Create				OUTPUT file-name	
	Read and Update				I-O file-name	
Disk	Extend	file-name	DISK-name	file-name LABEL RECORDS STANDARD	EXTEND file-name	file-name
Sort		file-name	DISK-name	SD file-name	N/A	N/A

Figure H-1 (Part 1 of 2). Sequential Files

Device	Function	Optional Entries					
		Environment Division			Data Division	Procedure Division	
		SPECIAL-NAMES	RESERVE	Other Clauses	FD Entry Clauses	Input/Output Verbs	Open Mode
Printers	Print	CSP[3] CO1	1 2	ACCESS SEQUENTIAL SAME AREA RERUN FILE STATUS	BLOCK CONTAINS RECORD CONTAINS LINAGE IS FOOTING LINES AT TOP LINES AT BOTTOM	WRITE record-name [FROM] [{BEFORE AFTER ADVANCING {identifier-2 integer mnemonic-name PAGE}] [{AT END-OF-PAGE EOP}]	Output
Disk	Read		1	ACCESS SEQUENTIAL	BLOCK CONTAINS RECORDS CONTAINS	READ file-name [INTO] AT END	Input
	Create		2	SAME AREA RERUN	DATA RECORDS	WRITE record-name[1] [FROM]	Output
	Read and Update			FILE STATUS OPTIONAL	VALUE OF CODE SET	READ file-name [INTO] AT END REWRITE record-name[2] [FROM]	I-O
Disk	Extend		1 2	ACCESS SEQUENTIAL SAME AREA RERUN FILE STATUS OPTIONAL	BLOCK CONTAINS RECORD CONTAINS DATA RECORDS VALUE OF CODE SET	WRITE file-name [FROM]	Extend
Sort		N/A	N/A	N/A	RECORD CONTAINS DATA RECORDS	MERGE SORT RELEASE/ RETURN	N/A

[1]Create
[2]Update
[3]The 3284 Printer cannot perform suppress printing

Figure H-1 (Part 2 of 2). Sequential Files

Organization	Required Entries						
	Environment Division				Data Division	Procedure Division	
	ACCESS IS	SELECT	ASSIGN System-name	KEY Clause	FD Entry	OPEN	CLOSE
RELATIVE	[SEQUENTIAL]	file-name	DISK-external-name	RELATIVE	File-name LABEL RECORDS STANDARD	Input file-name	file-name [LOCK]
						I-O file-name	
	RANDOM	file-name	DISK-external-name	RELATIVE	file-name LABEL RECORDS STANDARD	Input file-name	file- name [LOCK]
						Output file- name	
						I-O file-name	

Figure H-2 (Part 1 of 2). Relative Files

	Optional Entries			
	Environment Division	Data Division	Procedure Division	
Organization	Valid Optional Clauses	FD Entry Clauses	Input/Output Verbs	Open Mode
RELATIVE (Sequential)	SAME AREA RERUN FILE STATUS OPTIONAL	BLOCK CONTAINS RECORD CONTAINS DATA RECORDS VALUE OF CODE-SET	READ file-name [INTO] AT END	Input
			START file-name [KEY IS] INVALID KEY	
			READ file-name [INTO] AT END REWRITE record-name [FROM]	I-O
			START file- name KEY IS INVALID KEY	
RELATIVE (Random)	SAME AREA RERUN FILE STATUS	BLOCK CONTAINS RECORD CONTAINS DATA RECORDS VALUE OF CODE-SET	READ file-name [INTO] INVALID KEY	Input
			WRITE record-name [FROM][1] INVALID KEY	Output
			READ file-name [INTO] INVALID KEY WRITE record-name [FROM][2] INVALID KEY REWRITE record-name [FROM][3]	I-O

[1]Create and Add
[2]Add Only
[3]Update Only

Figure H-2 (Part 2 of 2). Relative Files

Organization	Required Entries						
	Environment Division				Data Division	Procedure Division	
	ACCESS IS	SELECT	ASSIGN System-name	KEY Clause	FD Entry	OPEN	CLOSE
INDEXED	[SEQUENTIAL]	file-name	DISK-external-name	RECORD	file-name LABEL RECORDS STANDARD	INPUT file-name	file-name [LOCK]
						OUTPUT file-name	
						I-O file-name	
	RANDOM	file-name	DISK-external-name	RECORDS	file-name LABEL RECORDS STANDARD	INPUT file-name	file-name [LOCK]
						I-O file-name	
	DYNAMIC	file-name	DISK-external-name	RECORD	file-name LABEL RECORDS STANDARD	INPUT I-O	file-name

Figure H-3 (Part 1 of 2). Indexed Files

Organization	Optional Entries			
	Environment Division	Data Division	Procedure Division	
	Valid Optional Clauses	FD Entry Clause	Input/Output Verbs	Open Mode
INDEXED (Sequential)	SAME AREA RERUN FILE STATUS	BLOCK CONTAINS m RECORDS RECORD CONTAINS	READ file-name [INTO] AT END	Input
		DATA RECORDS VALUE OF	START file-name [KEY IS] INVALID KEY	
	RESERVE 1 SAME AREA RERUN	CODE-SET	WRITE record-name [FROM][1] INVALID KEY	Output
			READ FILE-name [INTO] AT END REWRITE record-name [FROM][2] INVALID KEY	I-O
	SAME AREA RERUN		START file-name [KEY IS] INVALID KEY	
INDEXED (Random)	APPLY CORE-INDEX SAME AREA RERUN FILE STATUS	BLOCK CONTAINS m RECORDS RECORD CONTAINS DATA RECORDS VALUE OF CODE-SET	READ file-name [INTO] INVALID KEY	Input
			READ file-name [INTO] INVALID KEY WRITE record-name [FROM][3] INVALID KEY REWRITE record-name [FROM][3] INVALID KEY	I-O
INDEXED (Dynamic)	APPLY CORE-INDEX SAME AREA RERUN FILE STATUS	BLOCK CONTAINS m RECORDS RECORD CONTAINS DATA RECORDS VALUE OF CODE-SET	READ file-name NEXT RECORD READ file-name [INTO] INVALID KEY	Input or I-O
			REWRITE record-name [FROM] INVALID KEY	I-O

[1]Create
[2]Update
[3]Add

Figure H-3 (Part 2 of 2). Indexed Files

Required Entries					
Environment Division			Data Division	Procedure Division	
SELECT	ASSIGN system-name	ORGANIZATION	FD Entry	OPEN	CLOSE
file-name	WORKSTATION	TRANSACTION	file-name LABEL, RECORDS OMITTED	I-O file-name	file-name [WITH LOCK]

Figure H-4 (Part 1 of 2). TRANSACTION Files

Optional Entries			
Environment Division		Data Division	Procedure Division
SPECIAL- NAMES	FILE CONTROL	FD Entry Clauses	Input/Ouput Verbs
LOCAL-DATA ATTRIBUTE-DATA	FILE STATUS ASSIGN WORKSTATION- name-formats number-of-formats ACCESS SEQUENTIAL CONTROL AREA	RECORD CONTAINS DATA RECORDS Data Description Entry Boolean data item [INDICATOR]	ACQUIRE DROP READ file-name [INTO] [TERMINAL] [NO DATA] [AT END] WRITE record-name [FROM] [FORMAT] [TERMINAL] [STARTING] [ROLLING] [INDICATOR] ACCEPT FROM [FOR] DISPLAY UPON [FOR]

Figure H-4 (Part 2 of 2). TRANSACTION Files

Status Key 1	Status Key 2	Meaning
0		Successful completion
	0	No further information
	1	Initial READ from a REQUESTOR (IBM Extension)
1	0	At end of file (no outstanding invites)
2		Invalid key
	1	Sequence error
	2	Duplicate key when duplicates are not allowed
	3	No record found
	4	Boundary violation–indexed or relative file
3		Permanent error
	0	No further information
	4	Boundary violation–sequential file
9		Other errors (IBM Extensions)
	0	Invalid update, add, or output operation
	1	Undefined access type
	2	Logic error (I/O to unopened file, file locked, already OPEN, already CLOSED, or invalid operation)
	4	No current record pointer for I/O request
	5	Invalid or incomplete file information
	7	Invalid Op Code
	9	Undefined
	A	STOP requested by system operator
	C	Acquire operation failed, terminal not in standby mode
	D	Terminal operator released work station with INQUIRY key
	E	SRT program released its requestor, I/O rejected
	F	Acquire operation failed, either operator signed on is unauthorized or program is unauthorized to use resources
	G	Input data rejected, buffer too small
	H	Acquire operation failed, resource is unavailable or currently owned by another program
	I	Write operation failed, input data already received by Data Management
	N	Temporary error (error during session)

Figure H-5. Status Key Values and Meanings

Appendix I. Display Screen Format Specifications

S SPECIFICATIONS

Columns	Name	Entry	Explanation
1-5	Sequence number	Line number	Entry used to number the specification lines.
6	Form type	S	Identification for an S specification.
7		*	Asterisk in this column identifies this line as a comment line.
7-14	Format name	Display screen format name	Name of the display screen format that the $SFGR utility program creates from the S and D specifications.
15-16		Blank	
17-18	Start line number	01-24	Number of the line at which the display begins.
		V (column 17)	Start line number is determined by the user program.
19-20	Number of lines to clear	00-24	Number of lines to clear, including and following the starting line. The specified number of lines are cleared, beginning with the start line specified in columns 17 and 18.
21	Lowercase	Y	With the Shift key, operators key uppercase characters. Without the Shift key, operators key lowercase characters.
		N or blank	Operators key uppercase characters only.
22	Return input	Y or blank	Input fields on this display are returned to the user program, even if the operator enters no data.
		N	Input fields on this display are not returned to the user program unless the operator enters data in one or more of the fields. Then all input fields are returned to the program.
23-24		Blank	
25-26	Sound alarm	Y (column 25)	The alarm sounds when this display appears.
		N (column 25) or blank	The alarm does not sound when this display appears.
		01-99	The alarm sounds when this display appears only if the specified indicator is on.

S SPECIFICATIONS (Continued)

Columns	Name	Entry	Explanation
27	Enable function keys	Y	The function control keys identified by numbers in the key mask entry (columns 64 through 79) are enabled (allowed). If the key mask entry contains no numbers, all function control keys are disabled.
		N	The function control keys identified by numbers in the key mask entry are disabled (not allowed). If the key mask entry contains no numbers, all function control keys are enabled. If the operator presses a disabled function control key, an error message is displayed. The operator can then press the Error Reset key, followed by the correct function key.
		R	The function control key mask that is active for the display station is retained when this format is displayed.
		Blank	All function control keys are enabled. In this case, the key mask entry must not contain any numbers. *Note:* Function control keys that are not masked off and that are not supported by the program cause an error message to be displayed, which indicates that an invalid key was pressed.
28	Enable command keys	Y	The command keys identified by alphabetic characters in the key mask entry (columns 64 through 79) are enabled (allowed). If the key mask entry contains no alphabetic characters, all command keys are disabled.
		N	The command keys identified by alphabetic characters in the key mask entry are disabled (not allowed). If the key mask entry contains no alphabetic characters, all command keys are enabled. If the operator presses a disabled command key, an error message is displayed. The operator can then press the Error Reset key, followed by the correct command key.
		R	The command key mask that is active for the display station is retained when this format is displayed.
		Blank	All command keys are enabled. In this case, the key mask entry must not contain any alphabetic characters.

S SPECIFICATIONS (Continued)

Columns	Name	Entry	Explanation
29-30	Blink cursor	Y (column 29)	The cursor blinks when this display appears.
		N (column 29) or blank	The cursor does not blink.
		01-99	The cursor blinks only if the specified indicator is on.
31-32	Erase input fields	Y (column 31)	All unprotected input fields on the screen are erased, the keyboard is unlocked, and no output occurs. All D specifications are ignored. The use of Y is not recommended.
		N (column 31) or blank	The input fields are not erased.
		01-99	All unprotected input fields on the screen are erased and the keyboard is unlocked if the specified indicator is on.
33-34	Override fields	Y (column 33)	An override operation is performed. The use of Y is not recommended.
		N (column 33) or blank	The operation is not an override operation.
		01-99	An override operation is performed if the specified indicator is on. An override operation allows the screen to remain unchanged except for those fields that have indicators specified for them in columns 23 and 24 of the D specification, and those indicators are on. See *Special Display Format Considerations* in Chapter 7 for a description of an override operation. The record displayed by COBOL is exactly the same whether or not override is specified when the indicator in columns 23 and 24 of the D specification is on.
35-36	Suppress input	Y (column 35)	No input is returned to the user program until a format is displayed with suppress input specified as N or with the specified indicator off.
		N (column 35) or blank	Input is returned to the user program.
		01-99	Input to the user program is suppressed if the specified indicator is on.
37-63		Blank	

Columns	Name	Entry	Explanation
64-79	Key mask		The key mask is a string of numbers and/or alphabetic characters that identify keys to be enabled or disabled when this format is displayed. The key mask must begin in column 64 and cannot contain embedded blanks. The numbers and alphabetic characters can be intermixed. Numbers in the key mask identify function control keys: **Number** — **Function Control Key** 1 — Print 2 — ROLL↑ 3 — ROLL↓ 4 — Clear 5 — Help 6 — Record Backspace *Note:* COBOL recognizes only key numbers 2 and 3 (the ROLL keys). COBOL does not support key numbers 1, 4, 5, and 6. Alphabetic characters in the key mask identify command keys: **Alphabetic Character** — **Command Keys** A — 1 B — 2 C — 3 D — 4 E — 5 F — 6 G — 7 H — 8 I — 9 J — 10 K — 11 L — 12 M — 13 N — 14 P — 15 Q — 16 R — 17 S — 18 T — 19 U — 20 V — 21 W — 22 X — 23 Y — 24

D SPECIFICATIONS

Columns	Name	Entry	Explanation
1-5	Sequence number	Line number	Entry used to number the specification line.
6	Form type	D	Identification for a D specification.
7		*	Asterisk in this column identifies this line as a comment line.
7-12	Field name	Field name	Name of an input field, output field, or output/input field.
		Blank	This D specification line specifies only constant data.
13-14		Blank	
15-18	Field length	1-1919	The entry must be right-justified, but leading zeros are not required.
19-20	Line number	01-nn	Relative line number on which data appears. The actual line number is start line number (column 17 and 18 on the S specification) plus this line number, minus one. nn (maximum) = 24 - starting line number
21-22	Horizontal position	01-80	Column number of the first position of the field. Columns 19 through 22 cannot be 0101.
23-24	Output data	Y (column 23)	If constant data or a message identification code is also specified in columns 57 through 79, that constant data or the specified message is displayed in the field. If no constant data or message identification code is specified in columns 57 through 79, data from the user program output record is displayed.
		N (column 23) or blank	The field is not an output field.
		01-99	If the specified indicator is on when the format is displayed, data supplied by the user program is displayed in the field. If the specified indicator is off when the format is displayed, data specified in columns 57 through 79 is displayed. If no data is specified in columns 57 through 79, blanks are displayed. If the user program performs an override operation and the specified indicator is on, data supplied by the user program is displayed in the field. See *Special Display Format Considerations* in Chapter 7 for a description of the override operation. If the user program performs an override operation and the specified indicator is off, the field is unchanged.
25		Blank	

Columns	Name	Entry	Explanation
26	Input allowed	Y	The operator can enter information into the field from the keyboard.
		N or blank	The operator cannot enter information into the field from the keyboard.
27	Data type	A	The field can contain only alphabetic data.
		B or blank	The field can contain only alphameric data.
		K	The field can contain Katakana characters.
		N	The field can contain only numeric data. Commas, a period, a plus sign, or a minus sign can also be entered in this field.
		S	The field can contain only signed numeric data; the last position of the field is reserved for a sign. Only decimal digits (0 through 9) can be entered in the field. The field can be from 2 to 16 characters long.
28	Mandatory fill	Y	Operators must key all or key none of the field.
		N or blank	Operators can key all, none, or part of the input field.
			Note: Mandatory fill and adjust/fill (column 31) cannot be specified for the same field.
29	Mandatory entry	Y	Operators must enter at least one character or a blank in the input field.
		N or blank	Operators can bypass the input field.
30	Self check	T	The input field is a modulus 10 self-check field.
		E	The input field is a modulus 11 self-check field.
		Blank	The input field is not a self-check field.

Columns	Name	Entry	Explanation
31	Adjust/fill	Z	Information entered into the field is right-justified, and unused positions are filled with zeros.
		B	Information entered into the field is right-justified, and unused positions are filled with blanks.
		Blank	For signed-numeric fields, the information entered in the field is right-justified and blank fill is assumed. For all other fields, the information entered in the field is unchanged.
			Note: Mandatory fill (column 28) and adjust/fill cannot be specified for the same field.

Columns	Name	Entry	Explanation
32-33	Position cursor	Y (column 32)	Cursor appears at the first position of the input field when this format is displayed.
		N (column 32) or blank	Cursor does not appear at the first position of the input field.
		01-99	Cursor appears at the first position of the input field only if the specified indicator is on.
34	Enable Dup	Y	When the Dup key is pressed, the position of the cursor and the remainder of the field are filled with the duplicate character value (hex 1C), which is displayed as an asterisk (*). The duplicate characters must be processed by the user porgram.
		N or blank	The Dup key has no effect in the field.
35	Controlled field exit	Y	Cursor does not leave the input field until the operator presses a field exit key (Field Adv, Enter/Rec Adv, Field Exit, Field +, Field - [if the field is a signed-numeric field], Field Backspace, Home, Erase Input, or Dup).
		N or blank	Cursor automatically skips to the next unprotected field when the operator keys the last position of the field.
36	Auto record advance	Y	The input fields on the screen automatically return to the user program when one of the following occurs: • The operator enters the last character in the field. • The cursor is in the input field and the operator presses the Field Exit, Field +, or Field - key (if the field is a signed-numeric field).
		N or blank	Automatic record advance does not occur for this field.
37-38	Protect field	Y (column 37)	The cursor skips the field.
		N (column 37) or blank	The cursor does not skip the field.
		01-99	The cursor skips the field if the specified indicator is on. *Note:* If an override operation is used, this indicator is ignored.

Columns	Name	Entry	Explanation
39-40	High intensity	Y (column 39)	The field is displayed with high intensity.
		N (column 39) or blank	The field is displayed with normal intensity.
		01-99	The field is displayed with high intensity if the specified indicator is on.
			Note: High intensity, reverse image (columns 45-46), and underline (columns 47-48) cannot all be specified for the same field at the same time.
41-42	Blink field	Y (column 41)	The field blinks.
		N (column 41) or blank	The field does not blink.
		01-99	The field blinks if the specified indicator is on when the format is displayed.
43-44	Nondisplay	Y (column 43)	The field is nondisplay; that is, information in the field when the format is displayed or information entered into the field by the operator is not visible on the screen.
		N (column 43) or blank	The information in the field is displayed.
		01-99	The field is a nondisplay field if the specified indicator is on when the format is displayed.
45-46	Reverse image	Y (column 45)	The characters in the field appear as dark characters on a light background.
		N (column 45) or blank	The characters in the field appear as light characters on a dark background.
		01-99	The characters in the field appear as dark characters on a light background if the specified indicator is on when the format is displayed.
47-48	Underline	Y (column 47)	The field is underlined.
		N (column 47) or blank	The field is not underlined.
		01-99	The field is underlined if the specified indicator is on.

Columns	Name	Entry	Explanation
49	Column separators	Y	Each character position in the field is preceded by a column separator (a vertical line). The column separator does not require an additional character position.
		N or blank	Column separators are not used.
50-55		Blank	
56	Constant type	C	The constant information in columns 57 through 79 is to be displayed in the output field. C is required only if columns 57 through 79 are blank and you want to display all blanks in the field. C is invalid if an indicator is specified in columns 23 and 24.
		M	A message identification code and a message member identifier are entered in columns 57 through 79.
		Blank	If columns 57 through 79 contain constant information, that information is displayed. If columns 57 through 79 are blank, then information from the program output record area is displayed.
57-79	Constant data		This field specifies the information to be placed in an output or output/input field when the format is generated. If information is to be placed in the field, columns 57 through 79 should contain one of the following: • The actual information to be displayed. • A four-digit message identification code in columns 57 through 60 and a 2-character message member identifier in columns 61 and 62. *Notes:* 1. If columns 57 through 79 are blank and the field is an output field (Y in column 23), then information from the program output record is displayed. 2. If a message identification code is specified in columns 57 through 79, then only 6 bytes need be reserved for the field in the program output record area.
80	Continuation	X	If more than 23 characters of data are required, an X in column 80 indicates that the record is continued. Use columns 7 through 79 of the following record for the continued constant data. *Note:* A comment cannot follow a record with X in column 80.

Appendix J. Example of Conversion from Work Station PRPQ Support to Native COBOL TRANSACTION File Support

This appendix is provided to help demonstrate the difference between the Work Station PRPQ support for display stations and SSP-ICF sessions and the TRANSACTION file support provided by native COBOL. Figure J-1 is the listing of the screen formats that are used for both programs. Figure J-2 is the listing of the Work Station PRPQ program. Figure J-3 is the listing of the native COBOL TRANSACTION file program.

The major differences between the two programs are:

- The native COBOL program requires a TRANSACTION file specification in the ENVIRONMENT and DATA divisions, rather than a work station parameter list (WS-PARM-LIST).

- Subroutine CALLs are replaced by COBOL verbs.

For a detailed explanation of the COBOL language that replaces the work station subroutines, see Chapter 7, *TRANSACTION File Considerations and Sample Programs*.

```
    SZITEM      0124                03
    D           17 2 6Y                                    CENTER ITEM NUMBER
    DITEMNO      6 225   Y     Y   Y          Y
    D           13 233Y                                    C(OR / TO END)


  INPUT BUFFER DESCRIPTION

  FIELD                              START          END
  NAME              LENGTH           POSITION       POSITION

  ITEMNO               6                1              6



    SSHOWITEM    100                 N
    D             8 5 3Y                        Y          ITEM NO.
    D            22 514Y                       Y          DESCRIPTION
    D             9 541Y                       Y          PRICE
    D             6 553Y                       Y          ONHAND
    D             6 562Y                       Y          SOLD
    DITEMNO       6 6 3Y
    DONHAND       6 653Y
    DPENDING      6 662Y
    DPRICE        9 641Y
    DDESC        22 614Y
SYS-5077 W FORMAT CONTAINS FIELDS, PART OF A FIELD, OR AN ATTRIBUTE ON
         A LINE NOT CLEARED BY THE FORMAT.

  EXECUTION TIME OUTPUT BUFFER DESCRIPTION

  FIELD                              START          END
  NAME              LENGTH           POSITION       POSITION

  ITEMNO               6                1              6
  ONHAND               6                7             12
  PENDING              6               13             18
  PRICE                9               19             27
  DESC                22               28             49



    SZERROR     2400                N
    DMSG        14 1 2Y                           Y
SYS-5077 W FORMAT CONTAINS FIELDS, PART OF A FIELD, OR AN ATTRIBUTE ON
         A LINE NOT CLEARED BY THE FORMAT.

  EXECUTION TIME OUTPUT BUFFER DESCRIPTION

  FIELD                              START          END
  NAME              LENGTH           POSITION       POSITION

  MSG                 14                1             14
```

Figure J-1. Screen Formats for INQRM Sample Programs

```
 1  IDENTIFICATION DIVISION.
 2  PROGRAM-ID. INQRY1.
 3  ENVIRONMENT DIVISION.
 4  CONFIGURATION SECTION.
 5  SOURCE-COMPUTER. IBM-S34.
 6  OBJECT-COMPUTER. IBM-S34.
 7  INPUT-OUTPUT SECTION.
 8  FILE-CONTROL.
 9      SELECT DISK-FILE ASSIGN TO DISK-INV
            ORGANIZATION IS INDEXED
            ACCESS IS RANDOM
            RECORD KEY ITEMNO
            FILE STATUS IS DISK-RET-CODE.
10  DATA DIVISION.
11  FILE SECTION.
12  FD  DISK-FILE
        LABEL RECORDS ARE STANDARD.
13  01  DISK-REC.
14      05 ITEMNO            PIC X(6).
15      05 ONHAND            PIC X(5).
16      05 PENDING           PIC 9(5).
17      05 PRICE             PIC 9(5)V99.
18      05 DESC              PIC X(22).
19  WORKING-STORAGE SECTION.
20  77  DISK-RET-CODE        PIC XX.
21      88 RECORD-FOUND                  VALUE '00'.
22  01  WS-PARM-LIST.
23      05 FMT-LOAD-MEMBER   PIC X(8)    VALUE 'INQRYFMT'.
24      05 INPUT-LEN         PIC S9(4)   VALUE 49.
25      05 NUM-OF-INDS       PIC S99.
26      05 FMT-NAME          PIC X(8).
27      05 WSID              PIC XX.
28      05 START-LINE        PIC 99      VALUE 1.
29      05 CBL-RET-CODE      PIC XX.
30      05 AID-BYTE          PIC 99.
31      05 NO-ROLL-LINES     PIC 99.
32      05 TOP-ROLL-LINE     PIC 99.
33      05 END-ROLL-LINE     PIC 99.
34      05 SYS-RET-CODE      PIC X(4).
35      05 WORK-AREA         PIC X(4).
36      05 WS-RECORD.
37          10 ITEMNUM       PIC 9(6).
38          10 EOF-FLAG      REDEFINES ITEMNUM.
39              15 EOF       PIC X.
40                  88 END-OF-FILE       VALUE '/'.
41              15 FILLER    PIC X(5).
42          10 ONHAND        PIC Z(5)-.
43          10 PENDING       PIC Z(5)-.
44          10 PRICE         PIC ZZ,ZZZ.99.
45          10 DESC          PIC X(22).
```

Figure J-2 (Part 1 of 2). Work Station PRPQ Program

```
 46  01  ERR-MSG              PIC X(14)   VALUE
                              'ITEM NOT FOUND'.
 47  PROCEDURE DIVISION.
 48  000-CONTROL.
 49      PERFORM 100-HOUSEKEEPING.
 50      PERFORM 200-ACCEPT-INQRYS UNTIL END-OF-FILE.
 51      PERFORM 300-WRAP-UP.
 52      STOP RUN.
 53  100-HOUSEKEEPING.
 54      OPEN INPUT DISK-FILE.
 55      CALL 'WSOPEN' USING WS-PARM-LIST.
 56      CALL 'WSREAD'.
 57      MOVE 'ZITEM' TO FMT-NAME.
 58      CALL 'WSRITE' USING WS-RECORD.
 59      CALL 'WSREAD'.
 60  200-ACCEPT-INQRYS.
 61      MOVE ITEMNUM TO ITEMNO.
 62      READ DISK-FILE INTO WS-RECORD
 63          INVALID KEY PERFORM 250-REC-NOT-FOUND.
 64      IF RECORD-FOUND
 65          PERFORM 220-RECORD-FOUND.
 66      CALL 'WSREAD'.
 67  220-RECORD-FOUND.
 68      MOVE CORRESPONDING DISK-REC TO WS-RECORD.
 69      MOVE 'ZITEM' TO FMT-NAME.
 70      CALL 'WSRITE' USING WS-RECORD.
 71      MOVE 'SHOWITEM' TO FMT-NAME.
 72      CALL 'WSRITE' USING WS-RECORD.
 73  250-REC-NOT-FOUND.
 74      MOVE 'ZERROR' TO FMT-NAME.
 75      CALL 'WSRITE' USING ERR-MSG.
 76  300-WRAP-UP.
 77      CLOSE DISK-FILE.
 78      CALL 'WSCLOS'.

  PROGRAM SIZE = DATA DIVISION + PROCEDURE DIVISION + LITERALS + DTF/BUFFERS

     1776             173              503              76          1024


NO ERRORS DETECTED FOR THIS COMPILATION

  END OF COMPILATION
```

Figure J-2 (Part 2 of 2). Work Station PRPQ Program

```
 1  IDENTIFICATION DIVISION.
 2  PROGRAM-ID. INQRY.
 3  ENVIRONMENT DIVISION.
 4  CONFIGURATION SECTION.
 5  SOURCE-COMPUTER. IBM-S34.
 6  OBJECT-COMPUTER. IBM-S34.
 7  INPUT-OUTPUT SECTION.
 8  FILE-CONTROL.
 9      SELECT DISK-FILE ASSIGN TO DISK-INV
            ORGANIZATION IS INDEXED
            ACCESS IS RANDOM
            RECORD KEY ITEMNO
            FILE STATUS IS DISK-RET-CODE.
10      SELECT WORKSTN ASSIGN TO WORKSTATION-INQRYFMT-3
            ORGANIZATION IS TRANSACTION
            FILE STATUS IS CBL-RET-CODE, SYS-RET-CODE
            CONTROL-AREA IS WS-CONTROL.

11  DATA DIVISION.
12  FILE SECTION.
13  FD  DISK-FILE
        LABEL RECORDS ARE STANDARD.
14  01  DISK-REC.
15      05 ITEMNO              PIC X(6).
16      05 ONHAND              PIC 9(5).
17      05 PENDING             PIC 9(5).
18      05 PRICE               PIC 9(5)V99.
19      05 DESC                PIC X(22).
20  FD  WORKSTN
        LABEL RECORDS ARE OMITTED.
21  01  WS-RECORD.
22      05 ITEMNUM             PIC 9(6).
23      05 EOF-FLAG            REDEFINES ITEMNUM.
24          10 EOF             PIC X.
25              88 END-OF-FILE           VALUE '/'.
26          10 FILLER          PIC X(5).
27      05 ONHAND              PIC Z(5)-.
28      05 PENDING             PIC Z(5)-.
29      05 PRICE               PIC ZZ,ZZZ.99.
30      05 DESC                PIC X(22).
31  01  ERR-RECORD             PIC X(14).
32  WORKING-STORAGE SECTION.
33  77  DISK-RET-CODE          PIC XX.
34      88 RECORD-FOUND                  VALUE '00'.
35  01  WS-CONTROL.
36      05 AID-BYTE            PIC 99.
37      05 WSID                PIC XX.
38      05 FILLER              PIC X(8).
39  01  RET-CODES.
40      05 CBL-RET-CODE        PIC XX.
41      05 SYS-RET-CODE        PIC X(4).
42  01  ERR-MSG                PIC X(14)   VALUE
                               'ITEM NOT FOUND'.
```

Figure J-3 (Part 1 of 2). Converted Native COBOL Program

```
    /
 43  PROCEDURE DIVISION.
 44  000-CONTROL.
 45      PERFORM 100-HOUSEKEEPING.
 46      PERFORM 200-ACCEPT-INQRYS UNTIL END-OF-FILE.
 47      PERFORM 300-WRAP-UP.
 48      STOP RUN.
 49  100-HOUSEKEEPING.
 50      OPEN INPUT DISK-FILE, I-O WORKSTN.
 51      READ WORKSTN.
 52      WRITE WS-RECORD
             FORMAT IS 'ZITEM'.
 53      READ WORKSTN.
 54  200-ACCEPT-INQRYS.
 55      MOVE ITEMNUM TO ITEMNO.
 56      READ DISK-FILE INTO WS-RECORD
 57          INVALID KEY PERFORM 250-REC-NOT-FOUND.
 58      IF RECORD-FOUND
 59          PERFORM 220-RECORD-FOUND.
 60      READ WORKSTN.
 61  220-RECORD-FOUND.
 62      MOVE CORRESPONDING DISK-REC TO WS-RECORD.
 63      WRITE WS-RECORD
             FORMAT IS 'ZITEM'.
 64      WRITE WS-RECORD
             FORMAT IS 'SHOWITEM'.
 65  250-REC-NOT-FOUND.
 66      WRITE ERR-RECORD
             FROM ERR-MSG FORMAT IS 'ZERROR'.
 67  300-WRAP-UP.
 68      CLOSE DISK-FILE WORKSTN.

  PROGRAM SIZE = DATA DIVISION + PROCEDURE DIVISION + LITERALS + DTF/BUFFERS

      1902              145               504               76          1177


NO ERRORS DETECTED FOR THIS COMPILATION

  END OF COMPILATION
```

Figure J-3 (Part 2 of 2). Converted Native COBOL Program

abbreviated combined relational condition: A combined condition that omits a common subject, or a common subject and common relational operator, from a consecutive sequence of relation conditions.

access mode: A method used to read a specific logical record from, or to write a specific logical record to, a file assigned to an input/output device. Access can be sequential (records are referred to one after another in the order in which they appear on the file), it can be random (the individual records can be referred to in a nonsequential manner), or it can be dynamic (records can be accessed sequentially or randomly, depending on the form of the specific input/output request).

actual decimal point: The physical representation, using the decimal point character (. or ,), of the decimal point position in a data item. The actual decimal point appears in printed reports and requires a storage position in a data item. Contrast with *assumed decimal point*.

alphabet-name: A user-defined word, in the SPECIAL-NAMES paragraph, that names a character set and/or collating sequence.

alphabetic character: A character that is one of the 26 characters of the alphabet or a space. Alphabetic characters are uppercase.

alphanumeric character: Any character in the computer's character set.

alphanumeric edited character: An alphanumeric data item whose PICTURE character-string contains at least one B, 0, or 1.

American National Standard Code for Information Interchange (ASCII): The standard code used for information interchange between data processing systems, data communications systems, and associated equipment. The code uses a coded character set consisting of 7-bit coded characters (8 bits including parity check). The ASCII set consists of control characters and graphic characters.

American National Standards Institute (ANSI): An organization sponsored by the Computer and Business Equipment Manufacturers Association for the purpose of establishing voluntary industry standards.

ANSI: American National Standards Institute.

arithmetic expression: An arithmetic expression can be an identifier for a numeric elementary item, a numeric literal, such identifiers and literals separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

arithmetic operator: A symbol (1 character or a 2-character set) that indicates the arithmetic operation to be performed. Arithmetic operators include: + (addition), - (subtraction), * (multiplication), / (division), ** (exponentiation).

ascending key: The values by which data is ordered from the lowest value to the highest value of the key in accordance with the rules for comparing data items.

ascending key sequence: The arrangement of data in order from the lowest value of the key field to the highest value of the key field. Contrast with *descending key sequence*.

ASCII: American National Standard Code for Information Interchange.

assignment-name: A word that associates a file-name with an external device.

assumed decimal point: A logical decimal point position that does not occupy a storage position in a data item. It is used to align a value for calculations. Contrast with *actual decimal point*.

AT END condition: A condition that occurs at the following times:

- During the execution of a READ statement for a sequentially accessed file.
- During the execution of a RETURN statement when no next logical record exists for the associated sort or merge file.
- During the execution of a SEARCH statement when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

basic ideographic character set: An ideographic character set defined by IBM that contains 3707 characters consisting of 3226 Kanji characters and 481 additional characters which include Katakana, Hiragana, the alphabet (A through Z and a through z), numerics (0 through 9), roman numerals (I through X), Greek, Russian, and special symbols. The basic ideographic character set is defined in hardware for each ideographic-capable printer and display station.

binary item: A numeric data item that is represented internally in binary notation (that is, as a number in the base 2). Internally, each bit of the item is a binary digit with the sign as the leftmost bit.

block: A unit of data that is moved into or out of the computer. Synonymous with *physical record*.

Boolean data-type: A category of data items that are limited to a value of 1 or 0.

Boolean literal: See *literal*.

boundary violation: A condition that occurs when an output operation attempts to output more characters than is physically possible for a particular hardware device.

buffer: A portion of main storage into which data is read or from which it is written.

called program: A program that is the object of a CALL statement combined at object time with the calling program to produce a run unit.

calling program: A program that executes a CALL to another program.

character: One of a set of indivisible symbols that can be arranged in sequence to express information.

character set: All the valid COBOL characters.

character string: A sequence of characters that form a COBOL word, a literal, a PICTURE character-string, or a comment-entry.

checkpoint: A reference point in a program at which information about the program is recorded such that the program can be restarted at this reference point.

checkpoint records: Records that contain the status of a job and the system at the time the records are written by the checkpoint facility. These records provide the information necessary for restarting a job without having to return to the beginning of the job.

checkpoint record file: A disk file containing a collection of checkpoint records.

checkpoint restart: The process of resuming a job at a checkpoint after the job step terminated abnormally.

checkpoint/restart facility: A facility for restarting the execution of a program at some point other than the beginning, after the program was terminated due to a program or system failure. The restart begins at a checkpoint and uses checkpoint records to reinitialize the job.

class condition: A condition that states that the content of an item is all alphabetic or all numeric.

clause: An ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

COBOL: COmmon Business Oriented Language.

COBOL character set: The following 51 characters:

Character	Meaning
0, 1, . . ., 9	digit
A, B, . . ., Z	letter
	space (blank)
+	plus sign
-	minus sign (hyphen)
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol

collating sequence: The sequence in which the characters that are acceptable in a computer are ordered for sorting, merging, and comparing.

column: A character position within a print line. The columns are numbered from 1, by 1, starting at the leftmost character position of the print line and extending to the rightmost position of the print line.

combined condition: A condition that is the result of connecting two or more conditions with the AND or the OR logical operator.

comment: An annotation in the Identification Division or Procedure Division of a COBOL source program. A comment is ignored by the compiler. As an IBM extension, comments may be included at any point in a COBOL source program.

comment-entry: An entry in the Identification Division that is not translated by the compiler.

comment line: A source program line that is not translated by the compiler. The comment line can be used to document the program. A special form of the comment line can be used to cause page ejection before the comment line is printed.

Common Business Oriented Language (COBOL): A high-level programming language, similar to English, that is used primarily for commercial data processing.

compilation time: The time at which a COBOL compiler attempts to translate a source program into an object program.

compiler: A program that translates a program written in a high-level language into a machine language program.

compiler-directing statement: A statement that causes the compiler to take a specific action during compilation, rather than causing the object program to take a particular action during execution.

complex condition: A condition in which one or more logical operators (AND, OR, or NOT) act upon one or more conditions. Complex conditions include negated simple conditions, combined conditions, and negated combined conditions.

compound condition: A statement that tests two or more relational expressions. It may be true or false.

computer-name: A system-name that identifies the computer upon which the program is to be compiled or run.

condition: An expression in a program for which a truth value can be determined at execution time. Conditions include the simple conditions (relation condition, class condition, condition-name condition, switch-status condition, sign condition) and the complex conditions (negated simple conditions, combined conditions, negated combined conditions).

condition-name: A name assigned to a specific value, set of values, or range of values, within the complete set of values that a conditional variable can possess. Or, it is the name assigned to a status of an IBM-defined switch.

condition-name condition: A condition which states that the value of a conditional variable is a member of the set of values assigned to a condition-name associated with the conditional variable.

conditional expression: A simple condition or a complex condition specified in an IF, PERFORM, or SEARCH statement. (See *simple condition* and *complex condition.*)

conditional statement: A statement that causes the truth value of a condition to be determined and controls program execution depending on this truth value.

conditional variable: A data item, one or more values of which has a condition-name assigned to it.

CONFIGURATION SECTION: A section of the Environment Division of the COBOL program. It describes the overall specifications of computers.

connective: A word or a punctuation character that does one of the following:

- Associates a data-name, a paragraph-name, a condition-name, or a text-name with its qualifier
- Lines two or more operands in a series
- Forms a conditional expression

contiguous items: Consecutive elementary or group items in the Data Division that are contained in a single data hierarchy.

currency sign: The character $.

currency symbol: The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present, the currency sign is used. See *currency sign.*

current record: The record that is available in the record area associated with the file.

current record pointer: A conceptual entity that is used in sequential retrieval of the next record.

data clause: A clause that appears in a data description entry in the Data Division and provides information describing a particular attribute of a data item.

data description entry: A Data Division entry that describes the characteristics of a data item.

DATA DIVISION: One of the four main component parts of a COBOL program. The Data Division describes the files to be used in the program and the records contained within the files. It also describes any internal Working-Storage records that will be needed.

data hierarchy: The arrangement of data into levels, some of which are subordinate to others (a group item).

data item: A character or a set of contiguous characters (excluding literals in either case) defined as a unit of data by the COBOL program.

data-name: A user-defined word that names a data item. When used in the general formats, *data-name* represents a word that can be neither subscripted, indexed, nor qualified unless specifically permitted by the rules of that format. (See *identifier.*) An index-name is not a data-name.

debugging line: A COBOL statement executed only when the WITH DEBUGGING MODE clause is specified. Debugging lines can help determine the cause of an error.

debugging section: A Declaratives section that receives control when an identifier, file-name, or procedure-name is encountered in the Procedure Division.

declaratives: A set of one or more special-purpose sections, written at the beginning of the Procedure Division, that can be used for input/output error checking or debugging.

declarative-sentence: A compiler-directing sentence that specifies when a debugging section or an exception/error procedure is to be executed.

delimiter: A character or a sequence of contiguous characters that identifies the end of a string of characters and separates that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

descending key: The values by which data is ordered from the highest value to the lowest value of the key, in accordance with the rules for comparing data items.

descending key sequence: The arrangement of data in order from the highest value of the key field to the lowest value of the key field. Contrast with *ascending key sequence.*

digit: Any of the characters 0 through 9.

direct file: See *relative file.*

display screen: The part of a display station on which data, messages, or other information is displayed.

display screen format: A two-part table that defines a display presented by display station data management. Display screen formats are generated and placed in a library load member by the $SFGR utility program.

display station: An input/output device that contains a display screen on which data is displayed, and an attached keyboard from which data is entered. It can be used to request jobs or enter data. A display station can be designated as the system console or as a command or data display station at system configuration time.

display station local data area: A 256-byte area on disk that can be used to pass information between jobs and job steps during a session. A separate local data area exists for each command display station.

division: One of the four major parts in a COBOL program: Identification, Environment, Data, or Procedure.

division header: The reserved words and punctuation that indicate the beginning of one of the four divisions of a COBOL program.

dynamic access: An access mode in which records can be read from or written to a file in a non-sequential manner (see *random access*) and read from a file in a sequential manner (see *sequential access*) during the scope of the same OPEN statement.

EBCDIC: Extended binary-coded decimal interchange code.

EBCDIC character: Any one of the symbols included in the 8-bit EBCDIC (extended-binary coded decimal interchange code) set. All COBOL characters are included.

editing character: A single character or a fixed 2-character combination used to format output.

elementary item: A data item that is described as not being logically subdivided.

entry: Any descriptive set of consecutive clauses terminated by a period, written in the Identification, Environment, or Data Division of a COBOL source program.

ENVIRONMENT DIVISION: One of the four main component parts of a COBOL program. The Environment Division describes the computers upon which the source program is compiled and those on which the object program is executed; it also provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored.

execution time: The time at which the machine instructions of the object program are executed.

exponent: A number, indicating to which power another number (the base) is to be raised.

EXTEND mode: An open mode in which records are added to the end of a sequential file.

extended binary-coded decimal interchange code (EBCDIC): A set of 256 eight-bit characters.

external decimal item: See *zoned decimal item.*

figurative constant: A reserved word that represents a numeric or character value, or a string of repeated values. The word can be used instead of a literal to represent the value.

FILE-CONTROL: The name and header of an Environment Division paragraph in which the data files for a given source program are named and assigned to specific input/output devices.

file description entry: An entry in the File Section of the Data Division that contains information about the identification, the physical structure, and the record name of a file.

file-name: A name, associated with a file, defined in a file description entry or in a sort-merge file description entry.

file organization: The permanent file structure established at the time a file is created.

FILE SECTION: A section of the Data Division that contains descriptions of all externally stored data (or files) used in a program. Such information is given in one or more file description entries.

function-name: A name, defined by IBM, that identifies system logical units, system-supplied information, printer and card punch control characters, or program switches.

group item: A named set of contiguous elementary or group items.

hierarchy: A hierarchy is a set of entries that includes all subordinate entries to the next equal or higher level number.

IDENTIFICATION DIVISION: One of the four main component parts of a COBOL program. The Identification Division identifies the source program and the object program and, in addition, may include such documentation as the author's name, the installation where written, and the date written.

identifier: A data-name that is unique or is made unique by a combination of qualifiers, subscripts, or indexes.

ideographic character: A pictogram or graphic that requires 2 bytes of storage. Contrast with *alphanumeric character*.

ideographic character set: A character set that contains pictograms or graphics that can be used to represent ideas.

ideographic support: The combination of hardware and software elements that allow the use of ideographic data on the System/34.

imperative statement: A statement that specifies that an action is to be taken unconditionally. An imperative statement can consist of a series of imperative statements.

implementor-name: A system-name that identifies the external medium of a COBOL file and the name by which it is known to the system.

independent data item: A data item in the Working-Storage Section that has no relationship with other data items.

index: A computer storage position or register, the contents of which identify a particular element in a set of elements.

index data item: A data item in which the contents of an index can be saved.

index-name: A user-defined word that names an index. An index-name is not a data-name.

indexed data-name: A data-name, followed by one or more index-names enclosed in parentheses, that references an element or a set of elements in a table.

indexed file: A file with indexed organization.

indexed organization: The file structure in which each record is identified by the value of one or more keys within that record.

indicator: An internal switch that an object program uses to determine what to do when a particular event occurs.

input file: A file that is opened in the input mode.

input mode: An open mode where records can be read from the file.

INPUT PROCEDURE: A procedure that provides special processing of records when they are released to the sort function.

input-output file: A file that is opened in the I-O mode.

INPUT-OUTPUT SECTION: In the Environment Division, the section that names the files and external media needed by an object program. It also provides information required for the transmission and handling of data during the execution of an object program.

integer: A numeric data item or literal that does not include any character positions to the right of the decimal point. Where the term integer appears in formats, integer must be an unsigned numeric literal and must be nonzero unless the rules for that format explicitly state otherwise.

internal decimal item: See *packed decimal item.*

INVALID KEY condition: An execution-time condition in which the value of a key for an indexed or relative file does not give correct reference to the file (see input/output statements in the text of the manual for the specific error conditions involved).

Interactive Communications Feature (SSP-ICF): A feature of the SSP that includes interactive support for BSC and SNA communications as well as communications between programs within the system.

I-O mode: An open mode in which records can be read from, written to, or deleted from the file.

I-O-CONTROL: The name and the header for an Environment Division paragraph in which object program requirements for specific input/output techniques are specified. These techniques include rerun checkpoints, the sharing of same areas by several data files, and the use of a storage-resident cylinder index.

key: A data item that identifies the location of a record, or a set of data items that is used to place data in ascending or descending key sequence.

key word: A reserved word that is required by the syntax of a COBOL statement or entry.

language-name: A system-name that specifies a particular programming language.

level indicator: Two alphabetic characters, FD or SD, that identify the type of file description entry.

level number: A numeric character (1 through 9) or 2-character set (01 through 49, 66, 77, 88) that begins a data description entry, and establishes its level in a data hierarchy. Level-numbers 66, 77, and 88 identify special properties of a data description entry.

library-name: A user-defined word that names a library.

LINKAGE SECTION: A section of the Data Division that describes data made available from another program.

literal: A character string whose value is given by the characters themselves. For example, the numeric literal 7 has the value 7; the nonumeric literal "CHARACTERS", the value CHARACTERS; and the Boolean literal B"1", the value 1.

logical operator: A reserved word that defines the logical connection between conditions or negates a condition: OR (logical connective–either or both), AND (logical connective–both), and NOT (logical negation).

logical order: The order in which records are sequentially read from a file. For sequential and relative files, the logical order corresponds to the physical order of the records in the file. For indexed files, the logical order is based on the order of the keys in the index portion of the file.

logical record: The most inclusive data item. The level number for a logical record is 01.

main program: The highest-level program involved in a run unit.

main storage: Storage within the processing unit of the computer.

mass storage: A storage medium disk in which data can be collected and maintained.

mass storage file: A collection of records assigned to a mass storage device.

merge file: The temporary file that contains all the records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

mnemonic-name: A user-defined word associated with a function-name in the Environment Division.

mode: See *access mode.*

MRT: Multiple requestor terminal.

multiple requestor terminal (MRT) program: A program that can process requests from more than one requesting display station concurrently. Compare with *single requestor terminal (SRT) program.*

name: A word that defines a COBOL operand. A name is composed of not more than 30 characters.

native character set: The default character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

native collating sequence: The default collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

negated combined condition: The NOT logical operator immediately followed by a parenthesized combined condition.

negated condition: A condition whose truth value is negated by the NOT logical operator. If the condition is true, the negated condition is false. If the condition is false, the negated condition is true.

negated simple condition: The NOT logical operator immediately followed by a simple condition.

nest: To incorporate a structure or structures of some kind into a structure of the same kind. For example, to nest one loop (the nested loop) within another loop (the nesting loop); to nest one subroutine (the nested subroutine) within another suroutine (the nesting subroutine).

next executable sentence: The sentence to which control is transferred after execution of the current statement is complete.

next executable statement: The statement to which control is transferred after execution of the current statement is complete.

next record: The record that logically follows the current record of a file.

noncontiguous item: A data item in the Working-Storage Section of the Data Division that bears no relationship with other data items.

nonnumeric item: A data item that is alphanumeric, alphabetic, or Boolean.

nonnumeric literal: See *literal.*

numeric character: A character from the set of digits 0 through 9.

numeric edited item: A numeric item whose PICTURE character-string contains valid editing characters.

numeric item: An item whose contents must be numeric. If signed, the item can also contain a representation of an operational sign.

numeric literal: See *literal.*

OBJECT-COMPUTER: The name of an Environment Division paragraph in which the computer upon which the object program will be run is described.

object program: A set of instructions in machine-executable form. The object program is produced by a compiler from a source program.

object time: See *execution time.*

open mode: The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O, or EXTEND.

operand: The object of a verb or an operator; that is, an operand is the data or equipment governed or directed by a verb or operator.

operational sign: An algebraic sign, associated with a numeric data item or a numeric literal, that indicates whether the item is positive or negative.

optional word: A reserved word included in a specific format only to improve the readability of a COBOL statement or entry.

output file: A file that is opened in either the output mode or extend mode.

output mode: An open mode in which records can be written to a file.

OUTPUT PROCEDURE: A procedure that provides special processing of records when they are returned from the sort or merge function.

overflow condition: A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

overlay: To use the same area of storage for more than one procedure. See *Segmentation Feature* in Chapter 6.

packed decimal format: A format in which each byte (except the rightmost byte) within a field represents two digits, 0 through 9. The rightmost byte contains one digit and the sign. For example, the decimal value +123 is represented as 0001 0010 0011 1111. Contrast with *zoned decimal format.*

packed decimal item: A numeric data item that is represented internally in packed decimal format.

pad: To fill unused positions in a field with data, usually zeros or blanks.

paragraph: In the Procedure Division, a paragraph-name followed by a period and a space and by zero, one, or more sentences. In the Identification and Environment divisions, a paragraph header followed by zero, one, or more entries.

paragraph header: A reserved word, followed by a period and a space, that indicates the beginning of a paragraph in the Identification and Environment divisions.

paragraph-name: A user-defined word that identifies and begins a paragraph in the Procedure Division.

parameter: A variable or a literal that is used to pass data values between calling and called programs.

phrase: An ordered set of one or more consecutive COBOL character-strings that forms part of a clause or a Procedure Division statement.

physical record: A unit of data that is moved into or out of the computer. Synonymous with *block*.

procedure: One or more successive paragraphs or sections within the Procedure Division, which directs the computer to perform some action or series of actions.

PROCEDURE DIVISION: One of the four main component parts of a COBOL program. The Procedure Division contains instructions for solving a problem. The Procedure Division may contain imperative-statements, conditional statements, paragraphs, procedures and sections.

procedure-name: A paragraph-name or a section-name in the Procedure Division.

process: Any operation or combination of operations on data.

program-name: A user-defined word that identifies a COBOL source program.

pseudo-text: A sequence of character-strings and/or separators bounded by, but not including, pseudo-text delimiters. Pseudo-text is used in the COPY REPLACING statement for replacing text strings.

pseudo-text delimiter: Two contiguous equal signs (==) used to delimit pseudo-text.

punctuation character: Much as in English, a character used to separate COBOL elements or to identify a particular type of COBOL element: a comma, semicolon, period, quotation mark, left or right parenthesis, or space.

qualified name: A name that has been made unique by addition of one or more *qualifiers*.

qualifier: A name used to uniquely identify another name. Group data-names, section-names, and library-names can be used as qualifiers to form qualified names.

random access: An access mode in which specific logical records can be read from, written to, or deleted from a file in a nonsequential manner.

record: A set of one or more related data items that are grouped for processing. Records can be defined for an input/output device or for internal processing.

record area: A storage area in which a record described in a record description entry in the File Section is processed.

record description entry: The total set of data description entries associated with a particular record.

record key: A key whose contents identify a record within an indexed file.

record-name: A data-name for a record described in a record description entry.

relation character: One of the characters that expresses a relationship between two operands: = (equal to), > (greater than), < (less than).

relation condition: A condition that relates two arithmetic expressions and/or data items.

relational operator: A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used to construct a relation condition.

relative file: A file with a relative organization. Synonymous with *direct files*.

relative key: An unsigned integer data item that can be used directly by the system to locate a record in a file. Also known as relative record number.

relative organization: The file structure in which each record is uniquely identified by a positive integer value that specifies the record's ordinal position in the file.

reserved word: A predefined word used in a COBOL source program for syntactical purposes. It must not appear in a program as a user-defined name or as a system-name.

routine: A set of statements in a program that causes the computer to perform an operation or series of related operations.

run unit: A set of one or more object programs that function as a unit at execution time to provide a problem solution.

S/I control character: See *shift-in (S/I) control character*.

S/O control character: See *shift-out (S/O) control character*.

section: A set of zero, one, or more paragraphs or entries, called a section body, preceded by a *section header*. Each section consists of the section header and the related section body.

section header: A combination of words followed by a period and a space that indicates the beginning of a section in the Environment, Data or Procedure Division.

section-name: A user-defined word that names a *section* in the Procedure Division.

sector: A physical record on a storage device; it has a fixed length of 256 bytes.

sentence: A sequence of one or more statements; the last statement ends with a period followed by a space.

separator: A punctuation character used to delimit character-strings.

sequential access: An access mode in which records are read from, written to, or deleted from a file based on the logical order of the records in the file.

sequential file: A file with sequential organization.

sequential organization: The file structure in which the logical order of records in the file is determined by the order in which the records were first placed in the file.

sequential processing: The processing of logical records in the order in which records are accessed.

serial search: A search in which the members of a set are consecutively examined, beginning with the first member and ending with the last member.

SEU: Source entry utility.

shift-in (S/I) control character: A character that indicates the end of a string of ideographic characters. The shift-in control character is represented by hex 0F. Contrast with *shift-out (S/O) control character*.

shift-out (S/O) control character: A character that indicates the start of a string of ideographic characters. The shift-out control character is represented by hex 0E. Contrast with *shift-in (S/I) control character*.

sign condition: A condition that states that the algebraic value of a data item is less than, equal to, or greater than zero.

simple condition: Any single condition chosen from the set:

relation condition
class condition
condition-name condition
switch-status condition
sign condition

single requestor terminal (SRT) program: A program than can have only one requesting display station at a time. Contrast with *multiple-requestor terminal (MRT) program*.

sort file: The temporary file that contains all the records to be sorted by a SORT statement. The sort file is created and can be used only by the sort function.

sort-merge file description entry: An entry in the File Section that describes a sort file or a merge file.

SOURCE-COMPUTER: The name of an Environment Division paragraph describing the computer upon which the source program will be compiled.

Source Entry Utility (SEU): A portion of the Utilities Program Product that the operator uses to enter and update procedures and source programs in a library.

source program: A set of instructions that must be compiled before being executed.

special character: A COBOL character that is neither numeric nor alphabetic. Special characters in COBOL include the space ( ), and the period (.), as well as the following:

+ - * / = $, ") (; < >

special registers: Compiler-generated data items used to store information produced by specific COBOL features (for example, the DEBUG-ITEM special register).

special-character word: A reserved word that is an arithmetic operator or a relation character.

SPECIAL-NAMES: The name of an Environment Division paragraph and the paragraph itself in which names supplied by IBM are related to mnemonic-names specified by the programmer. In addition, this paragraph can be used to exchange the functions of the comma and the period or to specify a substitution character for the currency sign in the PICTURE string.

SRT: Single requestor terminal.

SSP: System Support Program Product.

SSP utility program: An SSP control program used by programmers in their daily system operations. For example, SSP utility programs can be used to copy files or initialize diskettes.

SSP-ICF: System Support Program Interactive Communications Feature.

SSP-ICF session: A logical information route between a System/34 application and a remote subsystem.

SSP-Interactive Communications Feature (SSP-ICF): See *Interactive Communications Feature.*

standard data format: The format in which data is described as to how it appears when it is printed, rather than how it is stored by the computer.

statement: A syntactically valid combination of words and symbols, beginning with a verb, that is written in the Procedure Division.

subject of entry: A data-name or reserved word that appears immediately after a level indicator or level number in a Data Division entry. It serves to reference the entry.

subprogram: A called program.

subscript: A positive integer whose value refers to a particular element in a table.

subscripted data-name: A data-name that has been made unique through the use of a subscript.

switch-status condition: A condition which states that a switch is currently on or off.

SYSTEM-CONSOLE: A COBOL function name associated with the operator's keyboard/display.

system-name: An IBM-defined name that has a predefined meaning to the COBOL compiler. System-names include computer-names, language-names, device-names, and function-names.

table: A set of logically consecutive data items that are defined in the Data Division by means of the OCCURS clause.

table element: A data item that can be referenced in a table.

test condition: A statement that taken as a whole, may be either true or false, depending on the circumstances existing at the time the expression is evaluated.

text-name: A user-defined word that identifies library text.

text-word: Any character-string or separator, except the space, in copied COBOL source or in pseudo-text.

TRANSACTION file: An input/output file used to communicate with work stations or for intersystem communications.

unary operator: A plus sign (+) or a minus sign (-), which precedes a variable or a left parenthesis in an arithmetic expression and which has the effect of multiplying the expression by +1 or -1, respectively.

UPSI: User program status indicator.

UPSI switch: A program switch that performs the functions of a hardware switch. Eight are provided: UPSI-0 through UPSI-7.

user-defined word: A word, required by a clause or a statement, that must be supplied by the user in a clause or statement.

user program status indicator (UPSI): One of a set of eight switches that can be set by and passed between application programs and procedures.

variable: A data item whose value can be changed during execution of the program.

verb: A COBOL reserved word that expresses an action to be taken by a COBOL compiler or an object program.

word: A character-string of not more than 30 characters, which forms a user-defined word, a system-name, or a reserved word.

work station: A device that lets a person transmit information to or receive information from a computer, or both, as needed to perform his job; for example, a display station or a printer.

WORKING-STORAGE SECTION: A section-name (and the section itself) in the Data Division. The section describes records and noncontiguous data items that are not part of external files but are developed and processed internally. It also defines data items whose values are assigned in the source program.

zoned decimal format: A format for representing numbers, in which the digit is contained in bits 4 through 7 and the sign is contained in bits 0 through 3 of the rightmost byte; bits 0 through 3 of all other bytes contain 1s (hex F). For example, in zoned decimal format, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. Contrast with *packed decimal format*.

zoned decimal item: A numeric data item that is represented internally in zoned decimal format.

IBM System/34
COBOL Reference Manual

READER'S COMMENT FORM

SC21-7741-5

Please use this form only to identify publication errors or to request changes in publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your nearest IBM branch office.

☐ If your comment does not need a reply (for example, pointing out a typing error) check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

☐ If you would like a reply, check this box. Be sure to print your name and address below.

Page number(s): Comment(s):

Please contact your nearest IBM branch office to request additional publications.

Name _____

Company or Organization _____

Address _____

_____

City State Zip Code

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

No postage necessary if mailed in the U.S.A.

Cut Along Line

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N. Y.

POSTAGE WILL BE PAID BY . . .

International Business Machines Corporation
Development Laboratory
Information Development, Department 532
Rochester, Minnesota 55901

Fold and tape | Please do not staple | Fold and tape

IBM

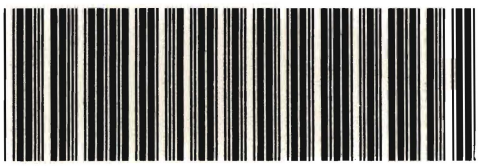
International Business Machines Corporation

IBM System/34 COBOL Reference Manual (File No. S34-24) Printed in U.S.A. SC21-7741-5

International Business Machines Corporation

IBM System/34 COBOL Reference Manual (File No. S34-24) Printed in U.S.A. SC21-7741-5

SC21-7741-05



SC21-7741-5